Summer 2014

Scalable Reasoning for Knowledge Bases Subject to Changes

Hui Shi
Old Dominion University

SCALABLE REASONING FOR KNOWLEDGE BASES SUBJECT TO CHANGES

by

Hui Shi
B.S. July 2003, Hefei University of Technology, China
M.S. June 2006, Hefei University of Technology, China

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
August 2014

Approved by:

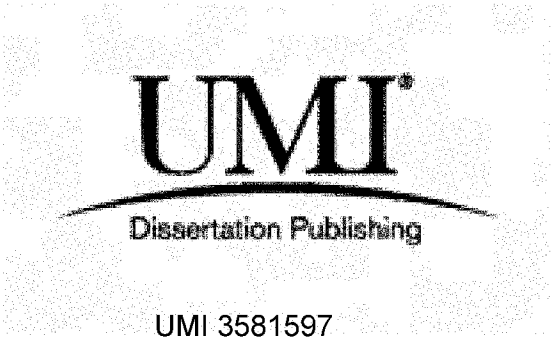
Kurt Maly (Co-Director)

Steven Zeil (Co-Director)

Xiaoping Liu (Member)

Mohammad Zubair (Member)

Harris Wu (Member)



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

SCALABLE REASONING FOR KNOWLEDGE BASES SUBJECT TO CHANGES

Hui Shi
Old Dominion University, 2014
Co-Directors: Dr. Kurt Maly
Dr. Steven J. Zeil

ScienceWeb is a semantic web system that collects information about a research community and allows users to ask qualitative and quantitative questions related to that information using a reasoning engine. The more complete the knowledge base is, the more helpful answers the system will provide. As the size of knowledge base increases, scalability becomes a challenge for the reasoning system. As users make changes to the knowledge base and/or new information is collected, providing fast enough response time (ranging from seconds to a few minutes) is one of the core challenges for the reasoning system.

There are two basic inference methods commonly used in first order logic: forward chaining and backward chaining. As a general rule, forward chaining is a good method for a static knowledge base and backward chaining is good for the more dynamic cases. The goal of this thesis was to design a hybrid reasoning architecture and develop a scalable reasoning system whose efficiency is able to meet the interaction requirements in a ScienceWeb system when facing a large and evolving knowledge base.

Interposing a backward chaining reasoner between an evolving knowledge base and a query manager with support of "trust" yields an architecture that can support reasoning in the face of frequent changes. An optimized query-answering algorithm, an optimized backward chaining algorithm and a trust-based hybrid reasoning algorithm are

three key algorithms in such an architecture. Collectively, these three algorithms are significant contributions to the field of backward chaining reasoners over ontologies.

I explored the idea of "trust" in the trust-based hybrid reasoning algorithm, where each change to the knowledge base is analyzed as to what subset of the knowledge base is impacted by the change and could therefore contribute to incorrect inferences. I adopted greedy ordering and deferring joins in optimized query-answering algorithm. I introduced four optimizations in the algorithm for backward chaining. These optimizations are: 1) the implementation of the selection function, 2) the upgraded substitute function, 3) the application of OLDT and 4) solving of the "owl:sameAs" problem.

I evaluated our optimization techniques by comparing the results with and without optimization techniques. I evaluated our optimized query answering algorithm by comparing to a traditional backward-chaining reasoner. I evaluated our trust-based hybrid reasoning algorithm by comparing the performance of a forward chaining algorithm to that of a pure backward chaining algorithm. The evaluation results have shown that the hybrid reasoning architecture with the scalable reasoning system is able to support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing both a fixed knowledge base and an evolving knowledge base.

This thesis is dedicated to my parents and my husband for their unconditional love, support and encouragement.

ACKNOWLEDGMENTS

I would never have been able to finish my dissertation without the guidance of my committee members and support from my parents and my husband.

I would love to express my deepest gratitude to my Ph.D. advisors, Dr. Kurt Maly and Dr. Steven J. Zeil, for their excellent guidance, advice, caring and patience. Thank you so much for believing in my abilities and thank you so much for your contributions of time and ideas to make these past five years a great experience for me.

I would love to thank my parents, for their unconditional and endless love and encouragement in all my efforts. Thank you for raising me to be happy, healthy, motivated. Thank you, Mom and Dad!

Finally, I would love to thank my husband, for always being my great partner. Thank you for your love, patience and support during my Ph.D. study.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Consider a potential chemistry Ph.D. student who is trying to find out what the emerging areas are that have good academic job prospects. What are the schools and who are the professors doing groundbreaking research in this area? What are the good funded research projects in this area? Consider a faculty member who might ask, "Is my record good enough to be tenured at my school? At another school?" Similarly consider an NSF program manager who would like to identify emerging research areas in mathematics that are not being currently supported by NSF. It is possible for these people each to mine this information from the Web. However, it may take a considerable effort and time, and even then the information may not be complete, may be partially incorrect, and would reflect an individual perspective for qualitative judgments. Thus, the efforts of the individuals neither take advantage of nor contribute to others' efforts to reuse the data, the queries, and the methods used to find the data. Qualitative descriptors such as "groundbreaking research in data mining" are likely to be accepted as meaningful if they represent a consensus of an appropriate subset of the community at large. Once accepted as meaningful, these descriptors can be realized in a system and made available for use by all members of that community. For example, "groundbreaking" research for one segment of the community could be work that results in many publications in refereed journals. For another segment it could be research that leads to artifacts that make people more productive, where "more productive" might mean to spend less time on finding papers in fields related to a given research problem. Qualitative descriptors also evolve

over time. For example, a community may later identify as another factor in "good research" that the degree to which it is "transformative". In addition to qualitative descriptors, useful queries may also be written using quantitative descriptors, for example: "What is the ordered list of PhD departments in CS when the ranking is the amount of research dollar spent in 2009?"

The system implied by these queries is an example of a semantic web where the underlying knowledge base covers linked data about science research that are being harvested from the Web and are supplemented and edited by community members. The query examples given above also imply that the system not only supports querying of facts but also rules and reasoning as a mechanism for answering queries.

In semantic webs, knowledge is formally represented by an ontology, as a set of concepts within a domain, and the relationships between pairs of concepts. The ontology is used to model a domain, to instantiate entities, and to support reasoning about entities (or facts). Ontologies represent facts as triples and we shall use the terms facts, triples, and instances synonymously. Common methods for implementing reasoning over ontologies are based on First Order Logic, which allows one to define rules over the ontology.

A number of projects (e.g., Libra [1, 2], Cimple [3], Arnetminer [4]) have built systems to capture limited aspects of community knowledge and to respond to semantic queries. However, all of them lack the level of community collaboration support that is required to build a knowledge base system that can evolve over time, both in terms of the knowledge it represents as well as the semantics involved in responding to qualitative questions. These systems are also homogeneous, in the sense that they harvest data from

one type of resources. A team at ODU is working on ScienceWeb [5, 6], which will combine diverse resources such as digital libraries for published papers, curricula vitae from the web, and agency data bases such as NSF's research grant data base and that will use collaboration as the fundamental approach to evolve its knowledge base.

1.1 ScienceWeb

In ScienceWeb the ODU team is developing a framework for a system that provides answers to qualitative and quantitative queries of a large evolving knowledge base covering science research. The system will support the community joining together, sharing the insights of its members, to evolve:

- the type of data to be gathered and how to organize them,
- the methods for collecting the data, their sources,
- the process of collecting them, and validating them,
- the meaning of qualitative descriptors and queries most needed and how they can be computationally realized.

ScienceWeb will need to scale to accommodate the substantial corpus of information about researchers, their projects and their publications. It will need to accommodate the inherent heterogeneity of both its information sources and of its user community. Finally, it must allow the semantic (qualitative) descriptors to evolve with time as the knowledge of the community grows and the problems the community researches change.

ScienceWeb will develop new tools, technologies, and a framework, allowing a community to: (a) collaboratively develop and evolve its domain knowledge base, (b) collaboratively develop queries for answering qualitative questions, and (c)

collaboratively help in automatically harvesting and validating information from different resources. In the long-term, this framework should prove useful in many domains and different contexts.

ScienceWeb is a platform where researchers including faculty, Ph.D. students and program managers can collaboratively work together to get answers of their queries from a consensus point of view or from their specific point of view. The collaborative aspect is not only in the construction of queries but in the construction of the underlying ontology, rules and instance data. The proposed architecture of the ScienceWeb is shown in Fig. 1.

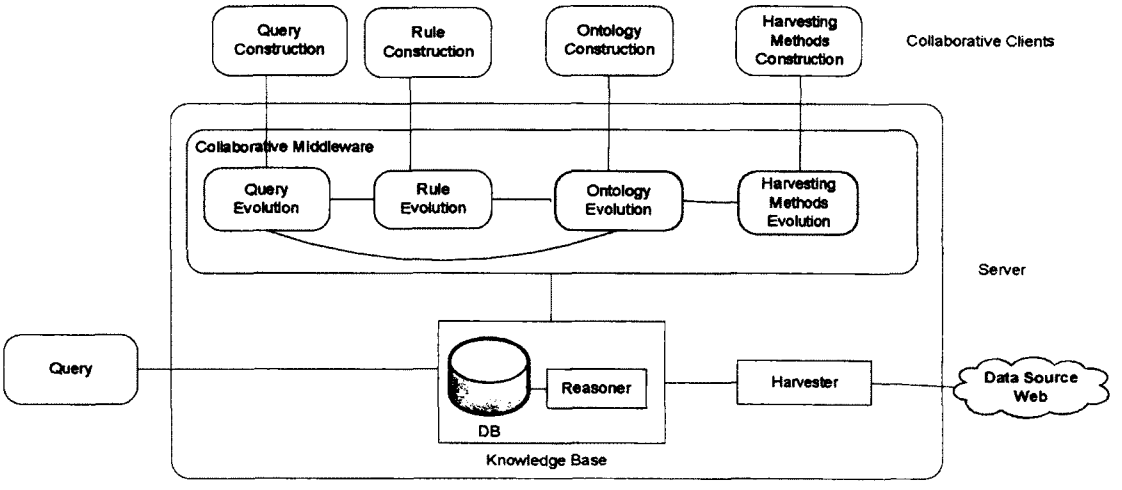


Fig. 1. Architecture of ScienceWeb

A traditional data mining architecture involves harvesting from data sources to populate a knowledge base, which in turn can then answer queries about the harvested content. This architecture is enhanced by adding a layer of collaborative clients for construction of queries, rules, ontological concepts, and harvesting methods, mediated by

a layer of server functions that oversee and support the evolution of each of those functional groups within the knowledge base.

The system is built, developed and evolved based upon users' collaborative contributions as shown in Fig. 2. Users contribute during querying & answering, harvesting and ontology evolution. Querying is not an ordinary job of posting, parsing and retrieving as in a conventional database system. Instead, it becomes an interactive, collaborative process. Harvesting and ontology evolution also benefit from the information provided by the users. Thus, collaboration is critical and widely spread throughout the system.

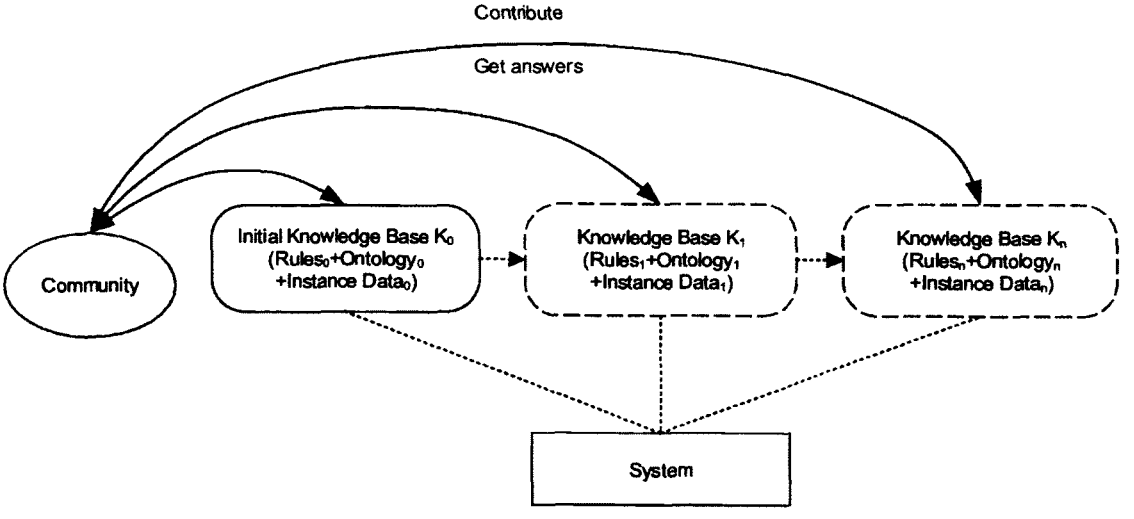


Fig. 2. Evolution of ScienceWeb

Queries and rules are the core of ScienceWeb system. How the queries and rules develop based on the collaboration is critical to the evolution of the system. Here are the main steps of development of queries and rules. First, users strive for consensus on rule

expression of qualitative descriptor and quantitative descriptor, consensus on adding rules into rule base, and consensus on changing the ontology. Second, when users construct queries and rules, the existing queries and rules from others' point of view should be made available for help and inspiration. Third, users will have access to methods for automatic query enrichment and rule prediction that are based on analysis of existing queries, rules and relationship among them, and analysis of behaviors and interests of specific subsets of a research community.

In ScienceWeb, a qualitative query is to be answered according to the criteria given in the form of custom rules by members of the community, which means inference from custom rules has to be done before returning the results of most queries. Current ontology reasoning systems do support inference from custom rules. But the current performance of inferencing from custom rules on large size instance data does not meet the requirement of real-time response. I have done experiments on different sizes of instance data conforming to the ScienceWeb ontology [5]. The experiment results explicitly show that when the size of the instance data grows from thousands to millions, the reasoning for a qualitative query with new custom rules takes minutes to hours, which is unacceptable for the real-time query-answering system [5]. It is critical to find approaches to solve the performance problem.

Some basic scenarios that illustrate the workings of the modules constituting ScienceWeb (as shown in Fig. 1) are:

Query construction and evolution:

- The user interacts with the Query Construction client to present his query.

- The Query Construction client works with the Query Evolution module and looks for similar past queries in the knowledge base.
- The user chooses to select one query from similar past queries, or to edit a past query, or to use the original query, unaltered.
- The query is passed on for query processing.

Query processing:

- The user submits a query, either a formerly constructed query or a new or modified one, arising from interaction with the Query Construction and Query Evolution modules.
- The reasoner and DB in the knowledge base work together to return the answers of the posted query.

Rule construction and evolution:

- The user constructs the rules (criteria) to express the qualitative descriptor and quantitative descriptor appearing in a new query for further inference.
- The user interacts with the Rule Construction client to introduce his rule.
- The Rule Construction client works with the Rule Evolution module to look for similar past rules to help users to compose rule.
- To compose a new rule, the user chooses to edit a past rule or to construct a totally new rule.
- After the new rule is constructed, the system informs the user that the new rule has been included in the rule base.

Ontology construction and evolution:

- The user works with the Ontology Construction client to add, edit or delete classes or properties in the ontology.
- The user interacts with the Ontology Evolution module to make sure that the new ontology is consistent after the changes.

Harvesting method construction and evolution:

- The user interacts with the Harvesting Method Construction client to describe the resources from which new instance data can be obtained.
- The harvester crawls some sample data from these resources.
- The Harvesting Method Evolution module verifies the validity of these resources.
- At a later time, the harvester obtains instance data within valid resources

1.2 Proposed Work

Collaboration is at the heart of the approach to build ScienceWeb. Such collaboration includes building and evolving the knowledge base, building, evolving and reusing queries and identifying, specifying methods and harvesting raw information. The interaction between the system and people must be effective enough to allow for collaborative development. Users are not willing to spend more than an hour, perhaps even only several minutes to wait for the response of the system when trying, for example, to create a new query [7].

Reasoning over the knowledge base provides support for answering qualitative questions and quantitative questions, whose scalability and efficiency influence greatly the response time of the system.

ScienceWeb is a system that collects various research related information. The more complete the knowledge base is, the more helpful answers the system will provide.

As the size of knowledge base increases, scalability becomes a challenge for the reasoning system. It may handle millions, even hundreds of millions, of items in the knowledge base. As users make changes to the basic descriptors of the knowledge base, providing fast enough response time (ranges from seconds to a few minutes) in the face of changes is one of the core challenges for the reasoning system.

The goal of this thesis is to design a hybrid reasoning architecture and develop a scalable reasoning system whose efficiency is able to meet the interaction requirements in a ScienceWeb system when facing a large and evolving knowledge base.

1.2.1 Architecture of an Adaptive Reasoning System for a Semantic Web

There are two basic inference methods commonly used in first order logic: forward chaining and backward chaining [8].

A question/answer system over a semantic web may experience changes frequently. These changes may be to the ontology, to the rule set or to the instances harvested from the web or other data sources. For the examples discussed in our opening paragraph, such changes could occur hundreds of times a day. Forward chaining is an example of data-driven reasoning, which starts with the known data in the knowledge base and applies modus ponens in the forward direction, deriving and adding new consequences until no more inferences can be made. Backward chaining is an example of goal-driven reasoning, which starts with goals from the consequents, matching the goals to the antecedents to find the data that satisfies the consequents. As a general rule, forward chaining is a good method for a static knowledge base and backward chaining is good for the more dynamic cases.

In order to achieve the goal of improving the performance and scalability of

reasoning, I introduce an adaptive reasoning architecture with a hybrid reasoner, a knowledge base and management modules. In this architecture, an adaptive mechanism is adopted to determine what part of the knowledge base is unaffected by changes and to switch between forward chaining and backward chaining depending on whether one or the other performs better. In the knowledge base, the storage will contain core facts obtained by the harvester, inferred instances from forward chaining, and standard reasoning rules as well as custom rules.

The resulting architecture of an adaptive reasoning system is presented in Fig. 3. To explain this architecture I present first a short description of the individual components and then a series of scenarios that will illustrate the sequence of modules executed and the resulting data flow upon various inputs from users.

Input from users: Queries and Changes

A query is the basic way for users to search and retrieve information. For example, in the query "Who are the groundbreaking researchers in Digital Library?" "groundbreaking" is a qualitative descriptor that has been evolved by one (or more) user(s) by developing custom rules; "researcher" and "digital library" are classes in the ontology.

Changes may be introduced to the ontology, to the custom rule set, or to instances as the harvested from the web. For example, people might not agree on the ontology as it was originally designed and they will make changes to reflect their own beliefs. Similarly, custom rules represent a personal understanding of qualitative descriptors. Custom rules will change as more people add their own opinion. Hopefully, these qualitative descriptors will evolve to a consensus. The collection of instances will be

enriched gradually with the discovery of new sources of information by individuals and the subsequent update of the methods of harvesting the information. Thus, changes of ontology, custom rule sets, and instances may occur with varying degree of frequency. Changes have a significant influence on the process of storage and performance of query no matter whether the query involves inferencing or not.

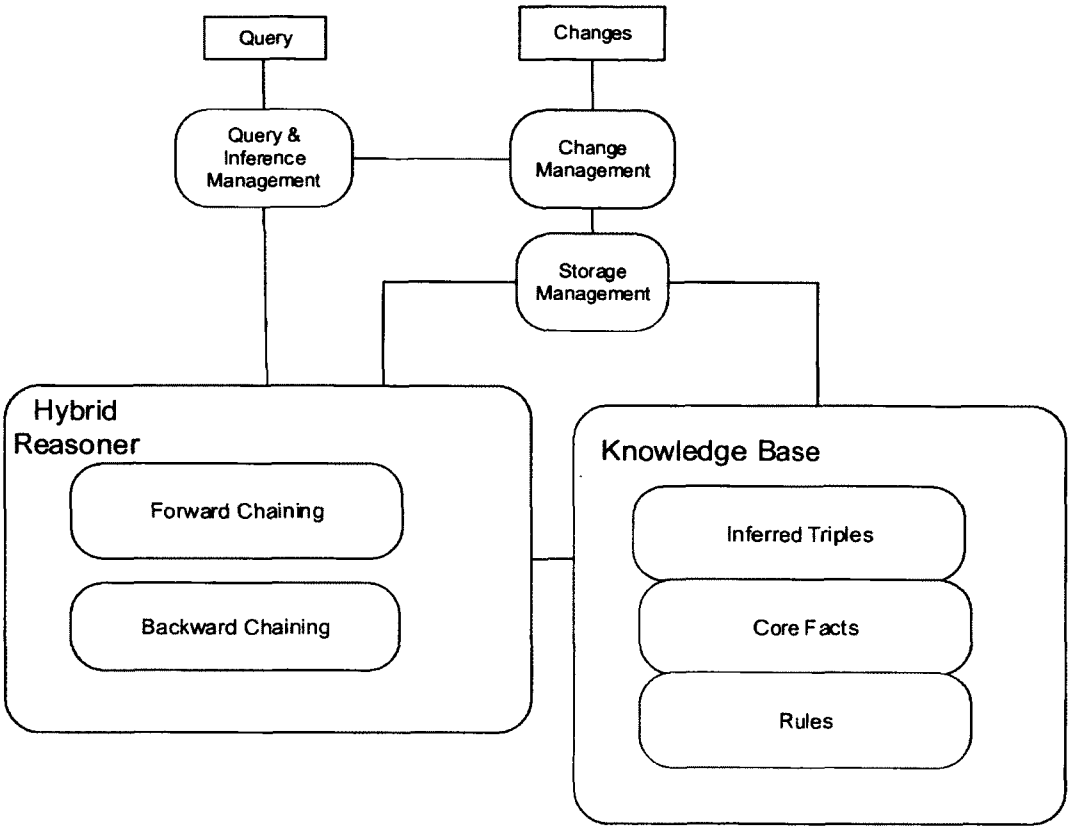


Fig. 3. Architecture of an Adaptive Reasoning System

Query & Inference Management

Query & Inference Management is the component that determines what part of the knowledge base is unaffected by the changes and that makes the choice between

Forward Chaining and Backward Chaining.

Change Management

Change Management records the history of all changes to the ontology, custom rules or instances. It not only provides change records to Query & Inference Management for the adaptive reasoning mechanism, but also communicates with the Storage Management module to realize the actual changes in the storage module.

Hybrid Reasoner

As a central component in the adaptive reasoning system, the Hybrid Reasoner is a combination of forward chaining and backward chaining that is responsible for the reasoning. Forward Chaining is the component that fires all of the rules in the system and generates all inferred triples at once – a process called materialization. After materialization, answering a query does not involve any reasoning but simple parsing, searching, and retrieving. Backward Chaining is the component that fires relevant rules in the system only during the processing of a query.

Storage Management

The Storage Management organizes storage to improve the scalability and performance of inferencing. This component provides mechanisms to group and index base triples obtained from the users and the harvester module and triples that have been inferred such that search and updates can be done efficiently.

Storage

There are three separate storage areas of data in the system: Inferred Triples, Core Facts, and Rules. Inferred Triples are generated by materialization, Core Facts include data instances and the ontology, and Rules include standard logic rules, custom rules

created at the start of the system and subsequent changes to these data.

1.2.2 Adaptive Reasoning Mechanism

ScienceWeb is a collaborative platform that integrates efforts from users to define what data are to be obtained in what way and how the data are to be organized and what forms the queries will be. After a bootstrapping process has generated an initial knowledge base, we expect frequent changes to all aspects of the knowledge base: ontology, rule set, harvesting methods, and instance data. It is one of my hypotheses, to be tested in the future, that changes to the ontology and rule set will stabilize over time whereas instance data will be continue to be changed as well as periodically harvested. The Adaptive Reasoning Mechanism is designed to select the appropriate reasoning method depending partially on the degree of change. Forward chaining is good in situations with infrequent or no updates. Queries, including qualitative queries, can then be executed without any inferencing. Fast response to queries without inference is a merit of forward chaining. Any update of the ontology, custom rules or instances requires reloading of data and inferencing all over again, resulting in slow responses to queries issued immediately subsequent to these changes. Backward chaining is good in situations with fast-changing data because backward chaining starts from the query then searches all the triples that meet the need, avoiding an entire materialization. As only rules and data related to the query are involved, answers can be returned within an acceptable time period. The critical question to be resolved in my thesis is how to delineate the impact of changes on the knowledge base so we know when and where to switch from one to the other reasoning methods.

1.3 Objectives

ScienceWeb is a platform where researchers including faculty, Ph.D. students and program managers can collaboratively work together to get answers of their queries from a consensus point of view or from their specific point of view. It will develop new tools, technologies, and a framework, allowing a community to: (a) collaboratively develop and evolve its domain knowledge base, (b) collaboratively develop queries for answering qualitative questions, and (c) collaboratively help in automatically harvesting and validating information from different resources.

I am addressing in this thesis only a select few of the challenging research issues ScienceWeb poses, and I am making certain assumptions about the context for the problem I will research. I will not address the collaborative aspects of ScienceWeb, including collaborative query, harvesting methods and ontology evolution.

In this thesis I will research the issues involved in designing a hybrid reasoning architecture and developing a scalable reasoning system whose scalability and efficiency are able to meet the requirements of query and answering in a semantic web system when facing both a fixed knowledge base and an evolving knowledge base. For evaluation purposes, I will develop a base query and rule set as well as instance data from a variety of sources. Specifically, the objectives are:

- Support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing a fixed knowledge base
    - Support custom rule reasoning to answer qualitative questions
    - Improve the scalability and efficiency of the backward chaining reasoner
    - Improve the scalability and efficiency of the query and answering process

  - Demonstrate completeness and soundness of the reasoning system
  - Demonstrate real-time or near real-time inferencing for a large knowledge base

- Support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing an evolving knowledge base
  - Classify and represent changes that the knowledge base faces
  - Introduce the concept of trust into the reasoning system
  - Develop a hybrid reasoning system that will combine forward chaining and backward chaining and adapt to changes in the knowledge base
  - Demonstrate completeness and soundness of the reasoning system
  - Demonstrate real-time or near real-time inferencing for a large knowledge base

In general, my main goal in this thesis is to support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing both a fixed knowledge base and an evolving knowledge base. For the purpose of demonstrating the performance of our system, I have run experiments on top of a widely used benchmark (up to 10 million facts). I have evaluated various optimizations that impact the effectiveness of the reasoning system as measured in response time. I have evaluated the system with respect to the scalability in terms of size of the knowledge base from thousands to 10 million facts. I have evaluated the completeness and soundness of the reasoning system.

CHAPTER 2

BACKGROUND

In this chapter, I will describe the background of the ScienceWeb project which provides the context for my thesis and that of adaptive reasoning for ScienceWeb which is the thesis's subject. The Related Areas section gives a general view of the context of my thesis area: digital libraries and the semantic web. The Thesis Area section focuses on the areas directly related to the work in this thesis, such as the background for reasoning and benchmarks for ontologies.

2.1 Related Areas

2.1.1 Scientific Literature Digital Libraries

Digital libraries are collections of digital objects. Digital libraries can be contrasted based on the source of their information such as scientific literature. The professional societies such as IEEE [9] or ACM [10] and commercial publishing houses such as Springer-Verlag [11] provide authoritative digital libraries for scientific publications. These libraries are limited in scope, being focused by design upon the publications of the sponsoring organization. These organizations maintain complete digital libraries for all their publications, but full access is typically restricted to paying members of the societies or subscribers. In addition to traditional search features, they often provide special features such as references cited by a paper, allow the exploration of the co-author relation, and sometimes provide citation counts.

A contrasting class of digital libraries consists of those systems that obtain their

content from the web. The stored content is typically a metadata record of a publication with references to the location of the actual document. Google Scholar [12] is designed to provide a simple way to search digital libraries of scholarly literature. One can search with keywords of common metadata of an article such as author, publication, date, and area. It provides a citation count, related papers and a listing of versions available for download. For articles in professional digital libraries, it provides metadata and a link to that digital library. DBLP [13, 14] provides bibliographic information on major computer science conference proceedings and journals. DBLP provides a faceted search on publication years, publication types, venues and authors. It also provides the functions of "Automatic Phrases" and "Syntactic Query Expansion" to generate more patterns and forms for entering keywords and key phrases. CiteseerX [15, 16], Libra [1, 2] and getCITED [17] are other systems that maintain and update their collection by continually crawling the web. All these systems have different levels of coverage. For instance, searching for "Smalltalk-80: the language and its implementation", CiteceerX does not include this book in its collection, Google scholar returns 979 as its citation count, Libra returns a citation count of 914, while getCITED returns 1. In some of these systems, community members can edit and correct harvested information.

2.1.2 Science-related Knowledge Bases

There are a large number of knowledge bases [3, 18-24] for a variety of domains. As an example, consider Cimple [3], which is being used to develop DBLife, a community portal for database researchers. In the Cimple project, researchers developed a knowledge base that draws its basic information from unstructured data on the web. It then analyzes the information and discovers relations such as co-authorship and

concentrates on having a consistent state of the information that it measures using precision and recall. It uses members of the community to correct and refine extracted information.

Another example of a sophisticated knowledge base in Computer Science is ArnetMiner [4], which provides profiles of researchers, associations between researchers, publications, co-author relationships, courses, and topic browsing. It has the capability to rank research and papers. It is a centrally developed system with fixed queries and schemas for data, but the knowledge base is continually growing as new data become available.

The Mathematics Genealogy Project [25] is a knowledge base that defines the transitive relation of advisor-PhD student for the domain of Mathematics. It is maintained by a few dedicated users with few automated tools. However, it has obtained significant coverage within its domain and even has expanded into Computer Science. Systems in this category are typically sustained by few people but most users are also contributors.

A number of these systems have developed interesting ways of harvesting information (converting unstructured or semi-structured information into structured), of forming natural language questions into formal queries, and of enhancing the precision, recall and efficiency when returning answers.

2.1.3 Ontologies

There has been an increasing effort in organizing web information within knowledge bases [26-29] using ontologies. Ontologies can model real world situations, can incorporate semantics which can be used to detect conflicts and resolve inconsistencies, and can be used together with a reasoning engine to infer new relations

or proof statements. The DBpedia [26] project focuses on converting Wikipedia content into structured knowledge. YAGO [27] is automatically derived from Wikipedia [30] and WordNet [31]. SwetoDblp is a large ontology derived from bibliographic data of computer science publications from DBLP. RKB Explorer [29] harvests information of people, projects, publications, research areas from a different types of resources. Ontology-driven applications also include data mining [32-35], software engineering [36], general natural language query systems [37, 38], and systems that help users build formal semantic queries [26, 39].

A number of tools exist for collaboratively designing and developing ontologies: CO-Protégé [40], Ontolingua [41], Ontosaurus [42], OntoEdit [43], WebODE [44], Kaon2 [45], OilEd [46]. With the exception of OilEd and Protégé they provide support for collaboration and conflict resolution [47].

Changes introduced to ontologies can result in structural and semantic inconsistencies, the resolution of which remains an open problem. Migration of instance data from one version of an ontology to the next is a critical issue in the context of evolving ontologies. Some ontology changes, such as creating new classes or properties, do not affect instances. When these changes occur, instances that were valid in the old version are still valid in the new version. However, other changes may potentially affect instances. In this latter case, some strategies can include having tools take their best guess as to how instances should be transformed, allowing users to specify what to do for a specific class of changes, or flagging instances that might be invalidated by changes.

2.1.4 Semantic Web

According to the W3C, "The Semantic Web provides a common framework that

allows data to be shared and reused across application, enterprise, and community boundaries." [48] The concept of a semantic web has been adopted in information retrieval to improve the accuracy of responses [49-54]. Assigning meanings to documents would keep documents expressing the same meaning with different words and eliminating documents expressing different meanings with the same words. For example, ontology-based information retrieval approaches [50, 51] aim to increase the quality of responses by capturing some semantics of documents. Another example is semantic indexing [52, 53], which aims to identify appropriate concepts that characterize the document content. Concepts from the semantic web have been adopted in information extraction to improve the accuracy of extraction [55-57]. For example, semantic-based text categorization aims to assign a class label to a document using semantic similarity [57]. The semantic web has been adopted in the biological sciences to make information accessible to researchers [58, 59]. The semantic web has been applied in corporate environments to facilitate the integration of information and provide support for decision making, for example, business processes [60], knowledge management [61]. A number of communities have adopted the semantic web as the basis for collaboration, for example, DBpedia [26], Friend of a Friend (FOAF) [62], Semantically-Interlinked Online Communities Project (SIOC) [63] and GoPubMed [64].

2.1.5 Sources of Information

Many of the systems already discussed can serve as a source of information for the knowledge base to be used in ScienceWeb. An open digital libraries such as DLBP can provide publications for computer science publications and, as it provides detailed citation records; it can also be used to build records of faculty and researchers and their

affiliations in the knowledge base. Information on funded research can be obtained from federal agencies in the United States such as NSF and NIH. NIH also requires all projects to publish their papers in PubMed [65] which can be used as well.

Significant sources of information will be the curricula vitae researchers typically publish on their websites. In computer science, a very high fraction of the researchers publish all relevant information about their research in their curricula vitae which are generally available to any crawler. More importantly most researchers regularly update these curricula vitae.

One of the few knowledge bases that provide information on the quality of an object is the study by the National Academy of Sciences [66] that ranks various departments and universities with respect to a number of criteria such as research output and student funding. It is interesting to note that, in a change to a similar study done ten years ago, this report allows now user to define their measure of quality and obtain dynamically rankings according to their view of quality.

2.2 Thesis Areas

2.2.1 Description Logic (DL)

Research in the field of knowledge representation and reasoning usually focused on methods for providing high-level descriptions of the world that can be effectively used to build knowledge-based systems. These knowledge-based systems are then able to get implicit consequences of their explicitly represented knowledge. Thus, approaches to knowledge representation are crucial to the ability of finding inferred consequences [67]. Early knowledge representation methods such as frames [68] and semantic networks [69] lack well-defined syntax and a formal, unambiguous semantics, which are elements of

qualified knowledge representation. Description logic was introduced into knowledge representation systems to improve the expressive power. Description logic [70] is a family of logic-based knowledge representation formalisms, which is designed to represent the terminological knowledge from an application domain [71].

There are many varieties of description logics. Different operators are allowed in these varieties. The expressive power is encoded in the label for a logic using the following letters[72]:

- $\mathcal{F}$: Functional properties.
- $\mathcal{E}$: Full existential qualification (Existential restrictions that have fillers other than `owl:thing`).
- $\mathcal{U}$: Concept union.
- $\mathcal{C}$: Complex concept negation.
- $\mathcal{S}$: An abbreviation for $\mathcal{ALC}$ with transitive roles.
- $\mathcal{H}$: Role hierarchy (subproperties - rdfs:subPropertyOf).
- $\mathcal{R}$: Limited complex role inclusion axioms; reflexivity and irreflexivity; role disjointness.
- $\mathcal{O}$: Nominals. (Enumerated classes of object value restrictions - `owl:oneOf`, `owl:hasValue`).
- $\mathcal{I}$: Inverse properties.
- $\mathcal{N}$: Cardinality restrictions (`owl:Cardinality`, `owl:MaxCardinality`).

- $\mathcal{Q}$: Qualified cardinality restrictions (available in OWL 2, cardinality restrictions that have fillers other than `owl:thing`).
- $(\mathcal{D})$: Use of datatype properties, data values or data types.

OWL is a DL-based language that is a standard ontology language for the Semantic Web. The design of OWL is based on the $\mathcal{SH}$ family of DLs. OWL 2 is based on the expressive power of $\mathcal{SROIQ}^{(\mathcal{D})}$, OWL-DL is based on $\mathcal{SHOIN}^{(\mathcal{D})}$, and for OWL-Lite it is based on $\mathcal{SHIF}^{(\mathcal{D})}$. [72]

A DL knowledge base consists of two parts: intentional knowledge (TBox), which represents general knowledge regarding a domain, and extensional knowledge (ABox), which represents a specific state of affairs. The "T" in term "TBox" denotes "terminology" or "taxonomy" which is built based on the properties of concepts and the subsumption relationships among the concepts in the knowledge. The "A" in term "ABox" denotes "assertional" knowledge that includes individuals of the specific domain. [67, 73]

2.2.2 Inference Methods

DL Reasoning Algorithm

The main reasoning tasks for DL reasoners are verifying KB consistency, checking concept satisfiability, concept subsumption and concept instances [71]. There are many algorithms for reasoning in DLs [71]. First are structural subsumption algorithms [74, 75], which first normalize the descriptions of concept, and then compare the syntactic structure of the normalized descriptions recursively. The disadvantage of this approach is that they are incomplete for expressive DLs, although they are efficient.

Second is the resolution-based approach [76-80], which transforms DLs into first-order predicate logic and then applies appropriate first-order resolution provers. Third is the automata-based approach [81-83], which is often more convenient for showing ExpTime complexity upper-bounds than tableau-based approaches. Fourth is the tableau-based approach [84], which is currently the most widely used reasoning algorithm for DLs. It had been previously used for modal logics [85], then introduced in the application of DLs by Schmidt-Schauß and Smolka [86] in 1991. This approach is able to deal with large knowledge bases from applications and is complete for expressive DLs. Furthermore, highly-optimized tableau-based algorithms [87] are proposed as the basis for the new Web Ontology Language OWL2.

Inference Methods in First Order Logic

There are three kinds of inference methods in First Order Logic (FOL), Forward chaining, Backward chaining and Resolution [8].

Forward chaining is an example of data-driven reasoning, which starts with the known data in the KB and applies modus ponens in the forward direction, deriving and adding new consequences until no more inferences can be made. Rete [88] is a well-known forward chaining algorithm. Backward chaining is an example of goal-driven reasoning, which starts with goals from the consequents matching the goals to the antecedents to find the data that satisfies the consequents. Resolution is a complete theorem-proving algorithm that proves the theorems by showing that the negation contradicts with the premises[8].

Materialization and Query-Rewriting

Materialization and query-rewriting are the most popular inference strategies

adopted by almost all of the state of the art ontology reasoning systems. Materialization means pre-computation and storage of inferred truths in a knowledge base, which is always executed during loading the data and combined with forward-chaining techniques. Query-rewriting means expanding the queries, which is always executed during answering the queries and combine with backward-chaining techniques.

Materialization and forward-chaining are suitable for frequent, expensive computation of answers with data that are relatively static. OWLIM [89, 90], Oracle 11g [91], Minerva [92] and DLDB-OWL [93] all implement materialization during loading of the data. Materialization permits rapid answer to queries because all possible inferences have already been carried out. But any change in the ontology, instances, or custom rules requires complete re-processing before responding to any new queries. Furthermore, a large amount of redundant data may be produced by materialization of a large knowledge base, which may slow the subsequent loading and querying.

Query-rewriting and backward-chaining are suitable for efficient computation of answers with data that are dynamic. Virtuoso [94], AllegroGraph [95] and Sher [96] implement dynamic reasoning when it is necessary. This approach improves the performance of answering new queries after data changes and simplifies the maintenance of storage. But frequent repeated queries in query-rewriting will require repeated reasoning, which is time-consuming compared to pure search in materialization. HStar [97] attempts to improve performance by adopting a strategy of partially materializing inference data instead of complete materializing.

A hybrid approach may give the best of both worlds. Jena [98] supports three ways of inferencing: forward-chaining, backward-chaining and a hybrid of these two

methods. In Jena's hybrid mode, any hybrid rules will be instantiated according to the variable bindings from the forward engine. Queries are answered using the backward engine by applying the merge of the supplied and instantiated rules to the data (raw data + inferred data). This hybrid approach has a fixed division between forward chaining and backward chaining. In my dissertation, I will explore an adaptive hybrid approach with a dynamic division between forward chaining and backward chaining for better performance under changing circumstances.

2.2.3 Combination Method of Ontologies and Rules

Classification of Combination

Ontologies are used to represent a domain of interest by defining concepts, recording relations among them and inserting individuals. Rules are mainly based on subsets of First Order Logic (FOL) and possible extensions. There is a trend to integrate ontologies and rules into the world of the Semantic Web.

The integration of ontologies and rules has been clearly reflected in Tim Berners-Lee's "Semantic Web Stack" diagram [99], which illustrates the hierarchy of languages that compose the architecture of the Semantic Web. There exist two reasons why to integrate ontologies and rules. First is expressive power: rules provide more expressive power to complement ontology languages, such as "composite properties". Second is reasoning techniques: existing considerable research on effective reasoning support of rules provide solid basis for ontology inferencing.

Antoniou et al. classify the integration of ontologies and rules according to the degree of integration [100], distinguishing between either the hybrid approach or the homogeneous approach as follows below. (Antoniou's use of the term "hybrid" is

unrelated to the hybrid reasoners discussed in the preceding section.)

In the homogeneous approach, the ontologies and rules are treated as a new logical language. There is no separation between predicates of an ontology and predicates of rules. Rules in this new language may be used to define classes and properties of an ontology. The inference in this approach is based on the inference of the new logical language. Examples of the homogeneous approach include [101-105]. Some of these are very popular in applications, such as DL+log [103], SWRL [104], DLP [105].

In the hybrid approach, there is a strict separation between predicates of ontologies and predicates of rules. Rules cannot define classes and relationships in the ontology, but some specific relations used in applications. The ontology reasoning and rule execution are performed by two different reasoners. Inferencing in the hybrid approach is based on the interaction between the ontology reasoner and the rule reasoner. There are also many examples of the hybrid approach AL-Log [106, 107], HD-rules [108], NLP-DL [109], CARIN [110], dl-programs [111], r-hybrid KBs [112],and DatalogDL [113].

RDF-entailment and RDFS-entailment are both examples of vocabulary entailment that capture the semantics of RDF and RDFS. Entailment rules are inference patterns that define what RDF triples [114] can be inferred from the existing knowledge[115]. Horst [116] defines "R-entailment" as an entailment over an RDF graph based on a set of rules. It is an extension of "RDFS entailment", extending the meta-modeling capabilities of RDFS. Horst defines pD* semantics in [117] as a weakened variant of OWL Full. Then in [118] pD* semantics were extended to apply to a larger subset of the OWL vocabulary. R-entailments incorporates pD* entailment. OWLIM [89,

90] is the practical reasoner based on OWL Horst [116-118]. Rule-based OWL reasoners are implemented based on the entailment rules, and the interpretation of these entailment rules relies on the rule engine.

There are three categories of semantic reasoners that adopt the above inference methods respectively. The first category is the DL reasoner, such as Racer [119], Pellet [120, 121], fact++ [122] and hermiT [123], with the inference of these reasoners being implemented by the popular tableau algorithm or hypertableau [123]. The second category contains reasoners based on the homogeneous approach, such as rule-based reasoner OWLIM [89, 90], OWLJessKB [124], BaseVISor [125], Jena [98, 126], Bossam [127], SweetRules [128], with the inference of these reasoners being based on the implementation of entailments in a rule engine. For example: Jena relies on one forward chaining RETE engine and one tabled Datalog engine [129]. OWLIM [89, 90] cannot work without support of TRREE. BaseVISor [125] and Bossam [127] are both RETE-based reasoners. The Datalog-driven reasoner Kaon2 [130, 131], which reduces a $\mathcal{SHIQ}^{(\mathcal{D})}$ KB to a disjunctive Datalog program, and the F-Logic based reasoner F-OWL [132] are both examples of homogeneous approach besides rule-based reasoners. The third category contains hybrid reasoners based on the hybrid approach of integration of ontology and rules as described in Section 2.2.3, such as AL-Log [106, 107], HD-rules [108], NLP-DL [109], CARIN [110], dl-programs [111], r-hybrid KBs [112], DatalogDL [113], DLE [133, 134] and Minerva [92] with the inference of these reasoners being implemented by integration of existing DL reasoner and with an existing rule engine.

DL reasoners have great performance on complex TBox reasoning, but they do not have scalable query answering capabilities that are necessary in applications with

large ABoxes [135-137]. Rule-based reasoners are based on the implementation of entailments in a rule engine. They have limited TBox reasoning completeness because they may not implement each entailment or they choose performance over completeness. TBox and ABox entailment rules are fired against ontological knowledge in Rule-based reasoners. Hybrid reasoners that integrate a DL reasoner and an existing rule engine can combine the strong points of both sides.

An example of such a hybrid reasoned is DLE [133, 134] which is a mixed framework combining the DL reasoning of a DL reasoner and the forward-chaining reasoning of a rule-based reasoner. By using the efficient DL algorithms, DLE disengages the manipulation of the TBox semantics from any incomplete entailment-based approach. And DLE also achieves faster application of the ABox related entailments and efficient memory usage[133]. However, there are three limitations in this framework:

1. It implements the materialization approach against the existing RDF graph based on the forward-chaining engine. With fast changing ontology and rules, DLE has to perform the materialization all over again, which makes some specific query and answering slower.
2. It does not implement the storage schema for more scalable ABox reasoning when the size of ABox exceeds that of main memory. It has only been tested in main memory, so the performance is limited due to scalability of rule-based reasoners.
3. It is based totally on the performance of the rule-based reasoner on firing the entailments without any optimization.

   It has also been mentioned in [133] that approaches described in [138] would

increase the performance. Minerva [92] is also a scalable owl ontology storage and inference system, which combines a DL reasoner and a rule engine for ontology inference, materializing all inferred results into a database. It implements the materialization strategy like DLE.

Support of Custom Rules

With custom rules, users can compose rules to introduce new concepts, such as the definition of a new property, into the structure of the TBox. Support of custom rules brings more flexibility into the reasoning systems. The combination of ontologies and rules makes the support of custom rules easier in terms of the expressive power and inference. In other words, the combination supports custom rules more naturally. SWRL is a good example. Some reasoners support custom rules by combining the logic and inferencing of SWRL to a differing extent, such as Racer, Pellet, hermiT, Bossam, SweetRules, Kaon2 and Ontobroker. SWRL is a system that follows the homogeneous approach of combination of ontologies and rules. Some reasoners support custom rules using their own rule formats based on the existing rule engine, such as Bossam, OWLIM, Jena, etc.

2.2.4 Scalability of Semantic Store

Storage Scheme

More and more semantic web applications contain large amounts of data conforming to an ontology. How to store these large amounts of data and how to reason with them becomes a challenging issue for the research in fields of Semantic Web and Artificial Intelligence and many semantic web applications. Research has been conducted on the storage scheme and the improvement of reasoning methods for large

size of data [139].

I will first discuss the storage scheme and its implementation of different triple stores.

Generally, there are two main kinds of triple stores: native stores (employing conventional file systems) and database-based stores, using relational or object relational databases.

Examples of native stores are OWLIM [89, 90], AllegroGraph [95], Sesame Native [140] ,Jena TDB [98, 126], HStar [97], Virtuoso [94].

AllegroGraph is a powerful graph-oriented database that can store pure RDF as well as any graph data-structure. The bulk of an AllegroGraph RDFstore is composed of assertions. Each assertion has five fields: subject (s), predicate (p), object (o), graph (g) and triple-id (i). The s, p, o, and g fields are strings of arbitrary size. AllegroGraph maintains a string dictionary which associates the unique string with a special number to prevent duplication. AllegroGraph creates indices which contain the assertions plus additional information to speed queries. AllegroGraph also keeps track of all the deleted triples. All these features result in fast speed for load and update.[141]

OWLIM has two versions. The "standard" OWLIM version, referred to as OWLIM-Lite, uses in-memory reasoning and query evaluation. The persistence strategy of this version is based on N-Triples. The indices of OWLIM-Lite are essentially hash tables. OWLIM-SE is an even more scalable not-in-memory version, which stores the contents of the repository (including the "inferred closure") in binary files with no need to parse, re-load and re-infer all the knowledge from scratch. OWLIM-SE uses sorted indices, which are seen as permanently stored ordered lists. OWLIM uses B+ trees to

index triples[139].

Jena TDB is a component of Jena for non-transactional native store and query. It is based on the file system. A TDB store consists of three parts, the node table, triple and quad indexes and the prefixes table. The node table generally stores the representation of RDF terms, and the node table provides two mappings. One is from Node to NodeId, and the other is from NodeId to Node. The default storage of the node table for the NodeId to Node mapping is a sequential access file, and that for Node to NodeId mapping is a B+ Tree. Triples are used for the default graph and stored as 3-tuples of NodeIds in triple indexes. Quads are used for named graphs and stored as 4-tuples. Each index has all the information about a triple. The prefixes table uses a node table and an index for GPU (Graph->Prefix->URI). And customized threaded B+ Trees are used in implementation of many persistent data structures in TDB [142].

In the HStar data model, OWL data has been divided into three categories. The first category contains OWL Class, OWL Property and Individual Resource. The second category contains the relation between elements in the first category. The third category contains characters defined on the OWL Property. HStar designs a customized storage model following the categorization of OWL data. It uses inner identifier OID instead of URI of an entity. HStar stores the mapping from the URI of an entity to its OID in global hash tables. Inheritance relations among Classes are stored in a tree structure Class Tree with a C-index for faster access to the data. The C-index is a B+ tree structure. Equivalence relations among Classes are stored in a B+ Tree and maintained in memory. Inheritance relations among Properties are stored in Property Tree. Equivalence relations among Classes are stored in a B+ Tree and maintained in memory. Inverse relations

among Properties are stored as data members of nodes in Property Tree. Therefore, HStar organizes its triples according to the categorization of different relationships and characteristics. Tree structures and lists are the main data structures that are used in the implementation of storage. B+ trees are also used to index triples [97].

RDF data are stored as quads, such as graph, subject, predicate and object tuples in Virtuoso. All such quads are in one table, which may require different indexing depending on the query. Bitmap indices are adopted by Virtuoso to improve the space efficiency. When the graph is known, the default index layout is a GSPO (graph, subject, predicate and object) as the primary key and OPGS as a bitmap index. When the graph is left open, the recommended index layout is SPOG for primary key, OPGS, GPOS and POGS as bitmap indices [143, 144].

Representative database-based stores are: Jena SDB [98], Oracle 11g R2 [91], Minerva [92], (Sesame + MySQL) [140], DLDB-OWL [93] and (Sesame+ PostgreSQL) [140]. They all take advantage of existing mature database technologies for persistent storage.

Comparing the native stores with database-based stores illustrates both their advantages and disadvantages. The advantage of native stores is that they reduce the time for loading and updating data. However, a disadvantage of native stores is that they are not able to make direct use of the query optimization features in database systems. Native stores need to implement the functionality of a relational database from the beginning, such as indexing, query optimization, and access control. As for database-based stores, the advantage is that they are able to make full use of mature database technologies, especially query optimization while the disadvantage is that they may be slower in

loading and updating data,

Reasoning over Large ABoxes

Due to the large size of instance data conformed to corresponding ontology in many knowledge bases, reasoning over large ABoxes has become an issue in the fields of semantic web and description logics. There are two main kinds of approaches to dealing with this issue. The first approach includes designing novel algorithms, schemes and mechanisms that enhance the reasoning ability on large and expressive knowledge bases. The second approach adopts simplification by reducing the expressive power of TBoxes describing large ABoxes.

In the first approach, Kaon2 has been shown to have better performance on knowledge bases with large ABoxes but with simple TBoxes [137] when compared with state-of-the-art DL reasoners such as Racer, FaCT++, and Pellet. One of the novel techniques adopted in the implementation of Kaon2 is reducing a SHIQ(D)knowledge base KB to a disjunctive Datalog program DD(KB) such that KB and DD(KB) entail the same set of ground facts. There are three advantages and extensions based on this transformation. First, existing research on techniques and optimizations of disjunctive Datalog programs could be reused, such as magic set transformation [145]. Second, DL-safe rules are combined with disjunctive programs naturally by translating them into Datalog rules, which increases the expressive power of the logic. Third, in [146] an algorithm is developed for answering conjunctive queries in $\mathcal{SHIQ}^{(\mathcal{D})}$ extended with DL-safe rules efficiently based on the reduction to disjunctive Datalog.

A scalable ontology reasoning method by summarization and refinement has been applied in Sher [96]. A summary of the ontology, called a "summary ABox", is used to

reduce the reasoning to a small subset of the original ABox, keeping the soundness and completeness [147]. The main idea of the method is to aggregate individuals that belong to the same concepts into the summary with all the necessary relations that preserve the consistency. Then queries are performed on the summary ABox rather than on the original ABox. Another process called refinement is used in determine the answers of the queries by expanding the summary ABox to make it more precise. Dolby et al. claimed that the method of summarization and refinement can also be treated as an optimization that any tableau reasoner can employ to achieve scalable ABox reasoning [148].

Guo et al. [149] have proposed a method that partitions a large and expressive ABox into small and comparatively independent components following the analyses of the TBox. Thus, specific kinds of reasoning can be executed separately on each component and the final results are completed through collecting and combining the results from each small component. After the partition, large ABoxes can be processed by state-of-the-art in-memory reasoners.

In [150] Wandelt and Möller present another example of the first approach using the method of "role condensates", which is a complete, but unsound approach to answer conjunctive queries in a proxy-like manner.

Using the second approach, Calvanese et al. [151, 152] have proposed a new Description logic, called DL-Lite, which is not only rich enough to capture basic ontology languages, but also requires low complexity of reasoning. The expressive power of this description logic allows conjunctive query answering through standard database technology so that the query optimization strategies provided by current DBMSs

can be used for the better performance. They concluded that this is the first result of polynomial data complexity for query answering over DL knowledge bases.

Horn-SHIQ [153, 154] is an expressive fragment of SHIQ and the Horn fragment of first-order logic. It provides polynomial algorithms for satisfiability checking and conjunctive query answering on large ABoxes.

2.2.5 Benchmarks

Benchmarks evaluate and compare the performances of different reasoning systems. A number of popular RDF and OWL benchmarks exist to conduct evaluation of performance over such variables as ABox size, TBox size, TBox complexity, and query response.

The SP2Bench SPARQL Performance Benchmark [155]has a scalable RDF data generator that creates DBLP-like data and a set of benchmark queries. SP2Bench has been applied to a number of well-known existing engines including the Java engine ARQ on top of Jena [156], SDB, Sesame and Virtuoso.

The Berlin SPARQL Benchmark(BSBM) [157] is also designed for comparison of the query performance of storage systems that expose SPARQL endpoints (i.e., a conformant SPARQL protocol service that enables users to query a knowledge base via the SPARQL language[158, 159]). BSBM simulates the realistic enterprise conditions and focuses on measuring SPARQL query performance against large amounts of RDF data. It is built around an e-commerce use case with a benchmark dataset, a set of benchmark queries and a query sequence. BSBM has been applied to a number of existing systems such as Virtuoso, Jena TDB, 4store [160], and OWLIM.

The Lehigh University Benchmark (LUBM) [161] is a widely used benchmark for

evaluation of Semantic Web repositories with different reasoning capabilities and storage mechanisms. LUBM includes an ontology for university domain, scalable synthetic OWL data, fourteen extensional queries and performance metrics.

The University Ontology Benchmark (UOBM) [162] extends the LUBM benchmark in terms of inference and scalability testing. UOBM adds a complete set of OWL Lite and DL constructs in the ontologies for more thorough evaluation on inference capability. UOBM creates more effective instance links by enriching necessary properties and improves the instance generation methods for more evaluation on scalability.

Both LUBM and UOBM have been widely applied to the state of the art reasoning systems to show the performance regarding different aspects[161, 162].

Although more and more systems consider the existing benchmarks as a standard way to evaluate and highlight their performance, some researchers have challenged the coverage of these benchmarks [163, 164]. Weithöner et al. [163] discussed some new aspects of evaluation such as ontology serialization, TBox complexity, query caching, and dynamic ontology changes. They concluded that there is still no current single benchmark suite that can cover both traditional and new aspects, as well as no reasoner that is able to deal with large and complex ABoxes in a robust manner. A set of requirement that is useful for the future benchmarking suites are given in the end.

2.2.6 Summary of Major Reasoning Systems

Jena

Jena is a Java framework for Semantic Web applications that includes an API for RDF. It supports in-memory and persistent storage, a SPARQL query engine and a rule-based inference engine for RDFS and OWL. The rule-based reasoner implements both

the RDFS and OWL reasoners. It provides forward and backward chaining and a hybrid execution model. It also provides ways to combine inferencing for RDFS/OWL with inferencing over custom rules. [98]

Pellet

Pellet is a free open-source Java-based reasoner supporting reasoning with the full expressive power of OWL-DL ($\mathcal{SHOIN}^{(D)}$) in description logic terminology). It is a great choice when sound and complete OWL DL reasoning is essential. It provides inference services such as consistency checking, concept satisfiability, classification and realization. Pellet also includes an optimized query engine capable of answering ABox queries. DL-safe custom rules can be encoded in SWRL. [121]

Kaon2

Kaon2 is a free (free for non-commercial usage) Java reasoner including an API for programmatic management of OWL-DL, SWRL, and F-Logic ontologies. It supports all of OWL Lite and all features of OWL-DL apart from nominal. It also has an implementation for answering conjunctive queries formulated using SPARQL. Much, but not all, of the SPARQL specification is supported. SWRL is the way to support custom rules in Kaon2. [131]

Oracle 11g

As an RDF store, Oracle 11g provides APIs that supports the Semantic Web. Oracle 11g provides full support for native inference in the database for RDFS, RDFS++, OWLPRIME, OWL2RL, etc. Custom rules are called user-defined rules in Oracle 11g. It uses forward chaining to perform inferencing. [91] In Oracle the custom rules are saved as records in tables.

OWLIM

OWLIM is both a scalable semantic repository and reasoner that supports the semantics of RDFS, OWL Horst and OWL 2 RL. OWLIM-Lite is a "standard" edition of OWLIM for medium data volumes. It is the edition in which all the reasoning and query are performed in memory. [90] Custom rules are defined via rules and axiomatic triples.

HermiT

HermiT is an efficient OWL reasoner based on a novel "hypertableau" reasoning algorithm. HermiT supports the semantics of OWL2. Since version 1.1, HermiT supports reasoning with DL Safe rules added to the ontology if there are no complex properties used in the rule bodies. [123]

2.2.7 Limitations of Existing Research

In order to implement an adaptive reasoning framework suitable for "ScienceWeb", an efficient, complete and scalable reasoning system with support of customs rules that can adapt to changes in the ontology, rules and instances is required.

As mentioned in Section 1.2.3, DL reasoners [119, 120, 122] have sufficient performance on complex TBox reasoning, but they do not have scalable query answering capabilities that are necessary in applications with large ABoxes. Rule-based OWL reasoners [89, 126] are based on the implementation of entailment rules in a rule engine. They have limited TBox reasoning completeness because they may not implement each entailment rule or they choose the performance instead of the completeness [133]. Hybrid reasoners [133, 165] that integrate a DL reasoner and a rule engine can combine the strong points of both sides. However, to our best knowledge, existing hybrid reasoners do not deal with evolving knowledge bases.

Forward-chaining and materialization, adopted in reasoning systems [89-93], is suitable for frequent, expensive computation of answers with data that are relatively static. However, any change in the ontology, instances or custom rules requires complete re-processing before response to the new queries. And large amount of redundant data will be produced for large knowledge base, which may slow the performance of loading and querying. Backward-chaining and query-rewriting, adopted in reasoning systems [94-96], is suitable for efficient computation of answers with data that are dynamic and infrequent queries. However, frequent, repeated queries will require repeated reasoning that is a waste of time. Reasonable adaptive hybrid approach would combine the strong points of both patterns for better performance under changing circumstances.

Persistent, external stores, either native stores or database-based stores, are necessary for a scalable reasoning system. Database-based stores may be slower in loading and updating data, but they are able to make full use of mature database technologies, especially, query optimization. As for native stores, they are not able to make use of the query optimization features in database systems. However, native stores reduce the time for loading and updating data greatly.

In this thesis I combine both forward-chaining and backward-chaining with support for persistent native stores for scalable reasoning. The purpose of this combination is to develop a scalable reasoning system whose scalability and efficiency is able to meet the requirements of query and answering in a semantic web system when facing both a fixed knowledge base and an evolving knowledge base.

CHAPTER 3

ONTOLOGY REASONING SYSTEMS USING CUSTOM RULES

In ScienceWeb we will be able to answer questions that contain qualitative descriptors such as "groundbreaking", "top researcher", and "tenurable at university x". ScienceWeb is being built using ontologies, reasoning systems and custom based rules for the reasoning system. In this chapter, I will address the scalability issue for a variety of supporting systems for ontologies and reasoning. In particular, I will discuss the impact of using custom inference rules that are needed when processing queries in ScienceWeb.

In this chapter, we evaluate the performance of Jena, Pellet, KAON2, Oracle 11g and OWLIM using representative custom rules, including transitive and recursive rules, on top of our ontology and LUBM [161]. In order to support custom rules, they do need to combine OWL inference and custom rule inference. These systems support custom rules in different formats and degrees. We will compare how the size of the ABox and the distribution of the ABox affect their inference performance for different samples of custom rules. We also compare query performance and query cache effects.

The remainder of this chapter is organized as follows: In Section 3.1 we discuss the ontology for ScienceWeb, a data generator for it and compare the generator to existing benchmarks. In Section 3.2 we evaluate various reasoning systems on multiple benchmarks on standard queries. In Section 3.3 we describe the data sets, custom rules and queries to be used in the experiments given in Section 3.4. Conclusions are given in Section 3.5.

3.1 Ontology Development and a Data Generator

3.1.1 Ontology Development

There have been a number of studies on reasoning systems using only their native logic. To provide credibility for our context, we used benchmark data from these studies, replicate their results with native logic, and then extend them by adding customized rules. I use LUBM [161] for comparing our results with earlier studies. The second set of data will emulate a future ScienceWeb. Since ScienceWeb at this time is only a plan, we need to use artificial data for experimentation.

Fig. 4 shows the limited ontology class tree we deem sufficient to explore the scalability issues. The ontology shown in Fig. 4 represents only a small subset of the one to be used for ScienceWeb. It was derived to be a minimal subset that is sufficient to answer a few select qualitative queries. The queries were selected to test the full capabilities of a reasoning system and to necessitate the addition of customized rules.

3.1.2 Synthetic Data Generator

In support of the performance analysis described above, a flexible system has been developed for generating a knowledge bases of varying sizes to serve as benchmarks [166]. This system is called "UnivGenerator". The major challenge for this generator is to not only produce ontology-conformant data sets of the desired size, but to guarantee a plausible distribution for the many properties that relate objects across the knowledge base. Existing generators such as LUBM [161] tend to work within an aggregation tree (e.g., for a university object generate a number of departments, for each department generate a number of faculty, for each faculty member generate a number of

papers authored by that faculty member). Obtaining reasonable distributions within such a tree is relatively straightforward. But when the domain expands to include other aggregation trees (e.g., for each publisher generate several journals, for each journal generate several papers) that must share objects with other aggregation trees, it is more of a challenge to maintain reasonable distributions for relations that span such trees (e.g., in how many different journals will a faculty member publish?) but such distributions are important to performance analysis of reasoning systems involving such relations.

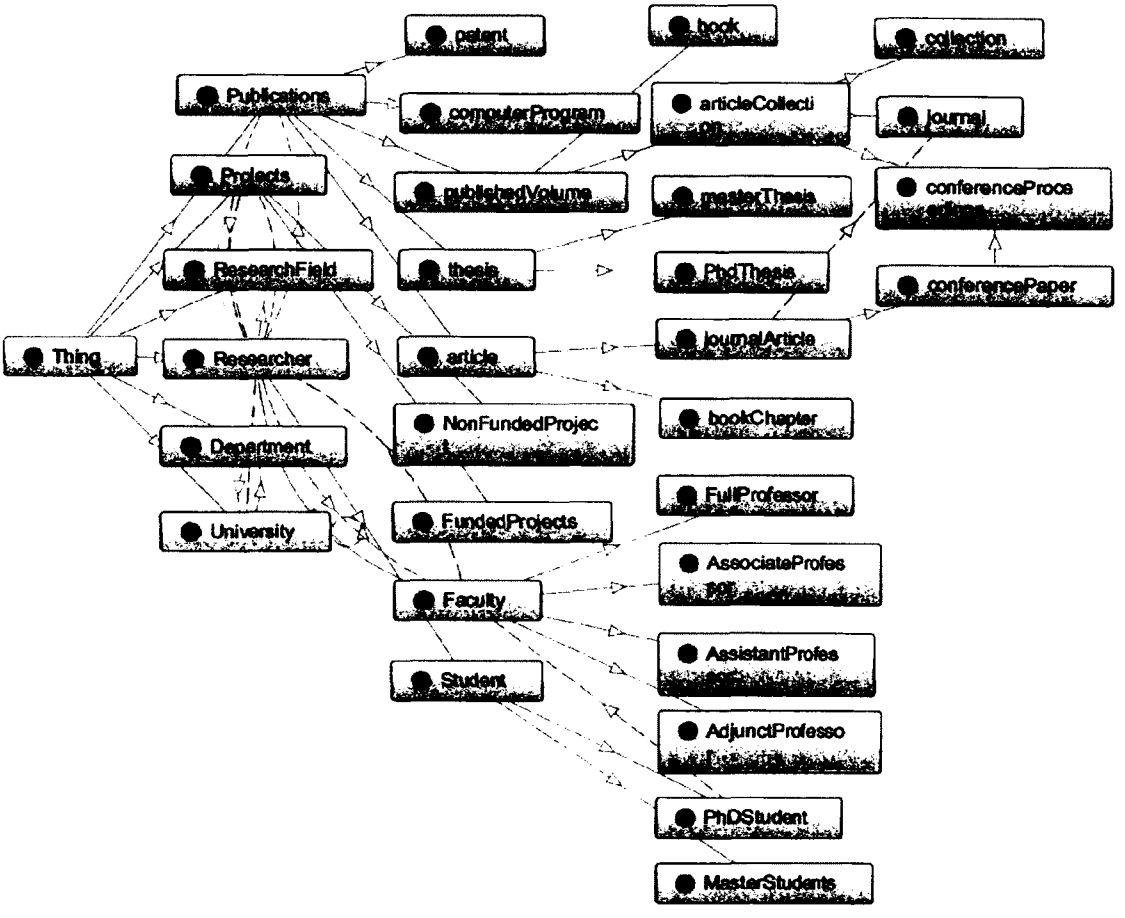


Fig. 4. Class tree of research community ontology

3.1.3 Comparison of the ScienceWeb, LUBM and UOBM Ontologies

The LUBM, the UOBM and the ScienceWeb ontologies are all about concepts and relationships in a research community. For instance, concepts such as Faculty, Publication, and Organization are included in all ontologies, as are properties such as `advisor`, `publicationAuthor`, and `worksFor`. All the concepts of the LUBM can be found in the ScienceWeb ontology, albeit the exact name for classes and properties may not be same.

ScienceWeb will provide more detail for some classes. For example, the ScienceWeb ontology has a smaller granularity when it describes the classification and properties of Publication. ScienceWeb provides more classes, for example, class `ResearchField` and class Projects. ScienceWeb also provides more properties for some classes, for example, `isRankedAt` and `hasProjectCount`, where `isRankedAt` is to hold rankings of universities, and `hasProjectCount` is to hold the number of projects of one university and one department. The addition of the concepts of research fields and projects, corresponding relationships and properties will make ScienceWeb more representative of the real research community.

LUBM starts with a university and generates faculty members for that university. The advisor of a student must be a faculty in the same university. The coauthors of a paper must be in the same university. In ScienceWeb we generate data that reflect a more realistic situation where faculty can have advisors at different universities and, co-authors can be at different universities.

UOBM extends LUBM in terms of inference and scalability testing. UOBM includes both OWL Lite and OWL DL ontologies for inference capability testing.

Moreover, UOBM generates links between individuals from different universities for scalability testing. Table I shows a comparison of the ScienceWeb, LUBM and UOBM including number of classes, properties and individuals per university. Following the rules of [162], the number of classes and properties used to define ABox are denoted in the bracket.

TABLE I
COMPARISON OF SCIENCEWEB, LUBM AND UOBM

Ontology+datasets	ScienceWeb	The LUBM	The UOBM	
			OWL Lite	OWL DL
No. of Classes	31(20)	43(22)	51(41)	69(59)
No. of Datatype Property	30(23)	7(3)	9(5)	9(5)
No. of Object Property	18(13)	25(14)	34(24)	34(24)
No. of Individuals in TBox	0	0	18	58
No. of Statements per University	3,400-7,000	90,000-110,000	210,000-250,000	220,000-260,000
No. of Individuals per University	300-500	8,000-15,000	10,000-20,000	10,000-20,000

Table I illustrates the size of LUBM ontology and how the LUBM synthetic data set scales up as we change the number of universities. ScienceWeb tends to generate classes and relationships that are more suggestive of a real research community. The ScienceWeb ontology in Table I is a limited ontology that represents only a small subset of the one to be used for ScienceWeb. It was derived to be a minimal subset that is sufficient to answer a few select qualitative queries.

3.2 Benchmark Study Using LUBM and UOBM

In this section, I replicate a benchmark study on LUBM and UOBM for the various reasoning systems under consideration. In the study, I employ the extensional

queries contained in the benchmarks themselves.

LUBM contains scalable synthetic OWL data. Different size of dataset can be generated for experiments. To identify the dataset, I use the following notation from paper [161] in the subsequent description: *LUBM (N, S)*: The dataset that contains N universities beginning at University0 and is generated using a seed value of S.

The instance generator in UOBM can create instances according to user specified ontology (OWL Lite or OWL DL). In addition, the user can specify the size of the generated instance data by setting the number of universities to be constructed. UOBM contains 6 test datasets for experiments set up for various goals: Lite-1, Lite-5, Lite-10, DL-1, DL-5 and DL-10. We use the following notation to identify these test datasets in the subsequent description: UOBM (Lite/DL-N): The dataset that contains N universities and is conformed to OWL Lite/OWL DL.

Due to scalability limits, some of the reasoning systems, such as Jena and Pellet, are not able to answer queries on large datasets, I therefore collected experimental results only on the small dataset LUBM (0, 1). These are presented in Table II. There are 14 extensional queries involved in this experiment.

In Table II, if there is a time-out (one hour), we mark it with an "n/a". Generally, most ontology reasoning systems have a reasonable performance except for Jena. Jena could not answer 7 out of 14 queries and its answer to query 6 is incomplete. Pellet does not return answers for query 2 and query 9. OWLIM has poor performance for query 7 and is not able to answer query 9 before time-out. Query 9 is more complicated than query 7. There are 2 constraints in query 7 while query 9 is featured by the most classes and properties in all the queries with 3 constraints. The complexity of a query does

influence the querying performance of OWLIM. Kaon2 and Oracle have the best overall performance.

TABLE II
QUERY PERFORMANCE ON LUBM (0, 1) UNIT: SECONDS

	Jena	Pellet	Oracle	Kaon2	OWLIM
q1	n/a	4.6	0.5	0.4	0.1
q2	n/a	n/a	0.9	0.5	0.1
q3	6.4	4.5	0.5	0.4	0.1
q4	354.2	4.5	1.5	0.6	0.1
q5	n/a	4.8	0.8	0.6	0.4
q6	6.9	5.3	1.5	0.6	2.9
q7	n/a	870	1.5	0.6	56.6
q8	n/a	684.1	2.7	1	7.3
q9	n/a	n/a	95.8	0.7	n/a
q10	n/a	n/a	0.8	0.6	0.1
q11	191	4.6	0.6	0.4	0.2
q12	6.2	5.8	0.7	0.6	0.1
q13	208.5	4.7	0.6	0.6	0.1
q14	6.8	5.3	1.4	0.4	1.7

The goal of our second experiment is to compare the reasoning systems in terms of query performance and completeness and soundness of the results using UOBM. Again, due to scalability limits, some reasoning systems are not able to answer queries on large datasets. I therefore only present the experiment results on the small dataset UOBM (DL-1). These are shown in Table III and Table IV. 15 extensional queries are involved in this experiment.

In Table III, if there is a time-out (one hour), we mark it with an "n/a". OWLIM is the only one system that could answer all the queries. The completeness and soundness of results are shown in Table IV. Oracle also performs well with a success on 14 out of 15 queries.

TABLE III
QUERY PERFORMANCE ON UOBM (DL-1) UNIT: SECONDS

	Jena	Pellet	Oracle	Kaon2	OWLIM
q1	n/a	9.9	1.8	n/a	0.2
q2	n/a	9.8	2	n/a	0.8
q3	n/a	10.2	0.8	n/a	0.4
q4	n/a	n/a	1.1	n/a	0.5
q5	n/a	9.9	0.7	n/a	0.1
q6	n/a	10.9	0.8	n/a	0.2
q7	n/a	10.3	0.7	n/a	0.2
q8	n/a	15.2	0.8	n/a	0.2
q9	n/a	9.6	1	n/a	0.4
q10	n/a	9.3	0.6	n/a	0.1
q11	n/a	1446.1	6.6	n/a	2.9
q12	n/a	11	1.3	n/a	0.4
q13	n/a	n/a	0.5	n/a	0.3
q14	n/a	1725.7	2.4	n/a	2.2
q15	n/a	14.3	0.4	n/a	0.1

TABLE IV
COMPLETENESS AND SOUNDNESS OF QUERY ON UOBM (DL-1)

	Jena	Pellet	Oracle	Kaon2	OWLIM
q1	n/a	32/32	32/32	n/a	32/32
q2	n/a	2512/2512	2512/2512	n/a	2512/2512
q3	n/a	666/666	666/666	n/a	666/666
q4	n/a	n/a	383/383	n/a	383/383
q5	n/a	0/200	200/200	n/a	200/200
q6	n/a	165/165	165/165	n/a	165/165
q7	n/a	19/19	19/19	n/a	19/19
q8	n/a	303/303	303/303	n/a	303/303
q9	n/a	0/1057	1057/1057	n/a	1057/1057
q10	n/a	24/25	25/25	n/a	25/25
q11	n/a	934/953	953/953	n/a	953/953
q12	n/a	65/65	65/65	n/a	65/65
q13	n/a	n/a	0/379	n/a	379/379
q14	n/a	0/6643	6643/6643	n/a	6643/6643
q15	n/a	0/0	0/0	n/a	0/0

3.3 Ontology Data, Custom Rule Sets and Queries

The UnivGenerator that I described in Section 3.1.2 generates a specified number of ontology triples within specific constraints. For instance, the user can specify that the number of publications of an associate professor ranges within 10-15 publications and the number of co-authors ranges from 0-4. The generator will ensure that triples are generated within these constraints and will make sure that the relations, e.g. co-author, are properly instantiated. The size range of the datasets in our experiments is listed in Table V. I generate 7 datasets for LUBM and 9 datasets for ScienceWeb, both ranges from thousand to millions for our experiment.

TABLE V
SIZE RANGE OF DATASETS (IN TRIPLES)

	Data set1	**Data set2**	**Data set3**	**Data set4**	**Data set5**
Science Web	3511	6728	13244	166163	332248
LUBM	8814	15438	34845	100838	624827
	Data set6	**Data set7**	**Data set8**	**Data set9**	
Science Web	1327573	2656491	3653071	3983538	
LUBM	1272870	2522900			

I now present the 5 rule sets and 3 corresponding queries that I will use in the next set of experiments. Rule sets were defined to test basic reasoning to allow for validation, such as co-authors having to be different, and to allow for transitivity and recursion. Rule sets 1 and 2 are for the co-authorship relation, rule set three is used in queries for the

genealogy of PhD advisors (transitive) and rule set 4 is to enable queries for distinguished advisors. Rule set 5 is a combination of the first 4 sets.

Rule set 1: Co-author

authorOf(?x, ?p)∧authorOf(?y, ?p)
⇒coAuthor(?x, ?y)

Rule set 2: validated Co-author

authorOf(?x, ?p)∧authorOf(?y, ?p)∧notEqual(?x, ?y)
⇒ coAuthor(?x, ?y)

Rule set 3: Research ancestor (transitive)

advisorOf(?x, ?y) ⇒ researchAncestor(?x, ?y)
researchAncestor(?x, ?y)∧researchAncestor(?y, ?z)
⇒ researchAncestor(?x, ?z)

Rule set 4: Distinguished advisor (recursive)

advisorOf(?x,?y)∧advisorOf(?x,?z)∧notEqual(?y,?z)
∧worksFor(?x,?u)
⇒ distinguishAdvisor(?x, ?u)
advisorOf(?x,?y)∧distinguishAdvisor(?y,?u)∧worksFor(?x,?d) ⇒
distinguishAdvisor(?x, ?d)

Rule set 5: combination of above 4 rule sets.

In the Jena rules language, these rules are encoded as:

```
@include <OWL>.
[rule1: (?x uni:authorOf ?p) (?y uni:authorOf ?p) notEqual(?x,?y)
->(?x uni:coAuthor ?y)]
[rule2: (?x uni:advisorOf ?y) -> (?x uni:researchAncestor ?y)]
[rule3: (?x uni:researchAncestor ?y)(?y uni:researchAncestor ?z)
->(?x uni:researchAncestor ?z)]
[rule4: (?x uni:advisorOf ?y) (?x uni:advisorOf ?z
notEqual(?y,?z) (?x uni:worksFor ?u) -> (?x
uni:distinguishAdvisor ?u)]
[rule5: (?x uni:advisorOf ?y) (?y uni:distinguishAdvisor ?u) (?x
uni:worksFor ?d) -> (?x uni:distinguishAdvisor ?d)]
```

In SWRL these rules are less compact. Rule 1 would be encoded as:

```
<swrl:Variable rdf:about="#x"/>
 <swrl:Variable rdf:about="#y"/>
 <swrl:Variable rdf:about="#p"/>
    <swrl:Imp rdf:about="rule1">
```

```
        <swrl:head rdf:parseType="Collection">
            <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate
rdf:resource="#coAuthor"/>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:resource="#y"/>
            </swrl:IndividualPropertyAtom>
        </swrl:head>
        <swrl:body rdf:parseType="Collection">
            <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate
rdf:resource="#authorOf"/>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:resource="#p"/>
            </swrl:IndividualPropertyAtom>
            <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate
rdf:resource="#authorOf"/>
                <swrl:argument1 rdf:resource="#y"/>
                <swrl:argument2 rdf:resource="#p"/>
            </swrl:IndividualPropertyAtom>
            <swrl:DifferentIndividualsAtom>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:resource="#y"/>
            </swrl:DifferentIndividualsAtom>
        </swrl:body>
    </swrl:Imp>
```

In OWLIM, rule1 would be encoded as:

```
Id: rule1
     x  <uni:authorOf> p                    [Constraint x != y]
     y  <uni:authorOf> p                    [Constraint y != x]
     -------------------------------
     x  <uni:coAuthor>  y                   [Constraint x != y]
```

I have composed 3 queries to use in these tests, expressed in SPARQL notation:

Query 1: Co-author

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
```

```
SELECT ?x  ?y
WHERE {?x uni:coAuthor ?y. ?x uni:hasName
\"FullProfessor0_d0_u0\" }
```

Query 2: Research ancestor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:researchAncestor ?y. ?x uni:hasName
\"FullProfessor0_d0_u0\" };
```

Query 3: Distinguished advisor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:distinguishAdvisor ?y. ?y uni:hasTitle
\"department0u0\" };
```

Queries are used with the rules sets that define the properties employed in the queries. Rule sets 1 and 2 are tested with query 1. Rule set 2 is tested with query 2. Rule sets 3 and 4 are tested with queries 2 and 3. Rule set 5 is tested with all queries.

3.4 Comparison of Reasoning Systems on Custom Rules

3.4.1 Experimental Environment and Metrics

The latest versions (as of the date of these experiments, May 2010) of OWL reasoning systems supporting custom rules have been chosen for the evaluation: Jena (2.6.2, 2009-10-22 release), KAON2 (2008-06-29 release), Pellet (2.2.0, 2010-07-06 release), OWLIM (3.3, 2010-07-22 release), and Oracle 11g R2 (11.2.0.1.0, 2009-09-01 release). As I wanted to insure that I was obtaining results that were commensurate with the earlier benchmark studies, I have taken the two main metrics from[167]. These are:

- Setup time: This stage includes loading and preprocessing time before any query can be made. This includes loading the ontology and instance data into the reasoning system, and any parsing, inference that needs to be done.

- Query processing time: This stage starts with parsing and executing the query and ends when all the results have been saved in the result set. It includes the time of traversing the result set sequentially. For some systems (Jena, Pellet, Kaon) it might include some preprocessing work.

All the tests were performed on a PC with a 2.40 GHz Intel Xeon processor and 16 G memory, running Windows Server 2008 R2 Enterprise. Sun Java 1.6.0 was used for Java-Based tools. The maximum heap size was set to 800M. I defined one hour as the time-out period.

3.4.2 Evaluation Procedure

Our goal is to evaluate the performance of different ontology reasoning systems in terms of reasoning and querying time using custom rules.

I am interested in two aspects of scalability. One aspect is simply the size of data. That is, I am interested in the performance of a system as the number of triplet's changes from small toy size to realistic sizes of millions. A second aspect of scalability I am interested is the expressive power of reasoning. That is, I am interested what some of the limits are in the type of questions ScienceWeb users can ask. To that end, I will perform the experiments with the transitive rules where we have different limits on the transitive chain.

Finally I am interested in the impact of using realistic models of the instance space (ScienceWeb) versus a simpler model (LUBM).

3.4.3 Comparison on Custom Rules

Setup Time

I begin by comparing setup times for the five systems under investigation. The

generated data sets contained thousands to millions of triples of information. The setup time can be sensitive to the rules sets used for inferencing about the data because some rules require filters (e.g., "not equal") and some rules express transitivity, or recursion. All will consequently involve different ABox reasoning with different sizes and distributions of ABoxes. Therefore, each system-ontology pair was examined using each of the 5 rules sets.

Jena, Pellet and Kaon2 load the entire data set into memory when performing inferencing. For these systems, therefore, the maximum accepted size of the ABox depends on the size of memory. In our environment, Jena and Pellet could only provide inferencing for small ABoxes (less than 600,000 triples). Kaon2 was able to return answers for ABoxes of up to 2 million triples. OWLIM and Oracle have better scalability because they are able to exploit external memory. They both are able to handle the largest data set posed, which is nearly 4 million triples. As the dataset grows, it turns out Oracle has the best scalability.

For smaller ABoxes (less than 2 million triples), Kaon2 usually has the best performance on setup time. However, as the size of the ABox grows past that point, only Oracle 11g and OWLIM are able to finish the setup before time-out occurs.

The performances of these systems vary considerably, especially when the size of dataset grows. OWLIM performs better than Oracle 11g on rule sets 1, 2 and 3. However, OWLIM is not good at rule set 4 with the ScienceWeb dataset. This appears to be because the ScienceWeb dataset involves more triples in the ABox inference for rule set 4 than does the LUBM dataset. Thus, when large ABox inferencing is involved in set-up, Oracle 11g performs better than OWLIM. Oracle 11g needs to set a "filter" when

inserting rule set 2 into the database to implement "notEqual" function in this rule set. As the ABox grows, more data has to be filtered while creating the entailment. This filter significantly slows the performance. For LUBM, Oracle 11g is not able to finish the setup for a one million triple ABox in one hour while OWLIM could finish in 100 seconds. Some data points are missing for larger size of datasets because we cut off experiments lasting longer than an hour.

Query Processing Time

Based on our results, OWLIM has the best query performance in most datasets in our experiments, but this superiority is overtaken as the size of dataset grows. Fig. 5 shows the time required to process query 1 on the ScienceWeb dataset.

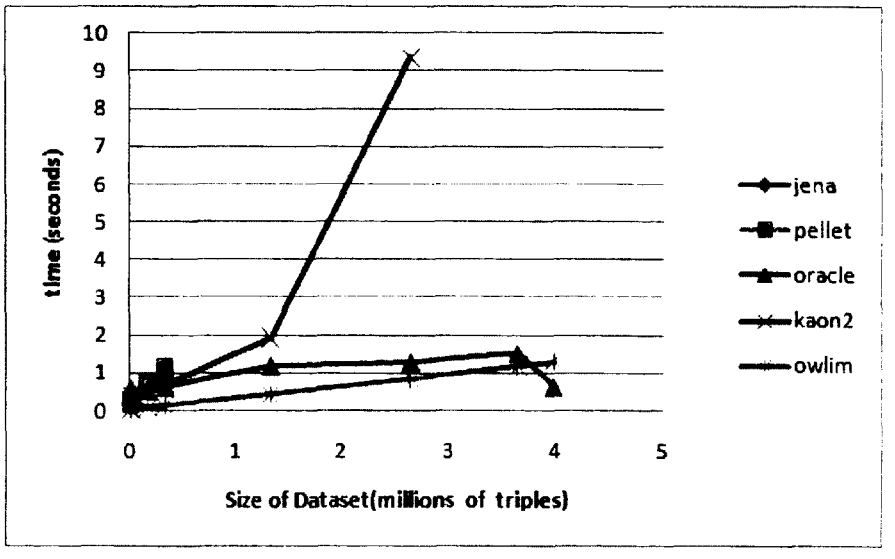


Fig. 5. Query processing time of query 1 for ScienceWeb dataset

In actual use, one might expect that some inferencing results will be requested repeatedly as component steps in larger queries. If so, then the performance of a

reasoning system may be enhanced if such results are cached.

Next, we therefore record the processing time for single queries and the average processing time for repeated queries. The average processing time is computed by executing the query consecutively 10 times. The average query processing time has been divided by the single query processing time to detect how caching might affect query processing. Table VI shows the caching ratio between processing time of one single query and average processing time on ScienceWeb ontology for query 1. Based on our results, OWLIM is the only one that does not have a pronounced caching effect. The caching effect of other systems becomes weaker as the size of dataset grows.

TABLE VI
CACHING RATIOS BETWEEN PROCESSING TIME OF SINGLE QUERY AND AVERAGE PROCESSING TIME ON SCIENCEWEB ONTOLOGY FOR QUERY

	Data set1	**Data set2**	**Data set3**	**Data set4**	**Data set5**	**Data set6**	**Data set7**	**Data set8**	**Data set9**
Jena	6.13	5.57	5.50	2.40	1.87				
Pellet	6.03	5.48	4.91	1.56	1.32				
Oracle	5.14	2.77	2.65	5.59	3.40	3.49	3.75	3.75	8.22
Kaon2	6.34	5.59	5.59	2.24	1.65	1.02	1.02		
OWLIM	1.83	1.83	1.30	1.48	1.20	1.05	1.07	1.01	1.03

3.4.4 Comparison among Systems on Transitive Rule

I anticipate that many ScienceWeb queries will involve transitive or recursive chains of inference, therefore, I next examine the behavior of the system on such chains of inference. Because the instance data for the LUBM and ScienceWeb ontologies may

not include sufficient long transitive chains, I created a group of separate instance files containing different number of individuals that are related via the transitive rule in rule set 3.

As Fig. 6 and Fig. 7 show, Pellet only provides the results before time-out when the length of transitive chain is 100. Jena's performance degrades badly when the length is more than 200. Only Kaon2, OWLIM and Oracle 11g could complete inference and querying on long transitive chains.

Kaon2 has the best performance in terms of setup time, with OWLIM having the second best performance. For query processing time, OWLIM and Oracle have almost same performance in terms of querying time. When the length of transitive chain grows from 500 to 1000, the querying time of Kaon2 has a dramatic increase.

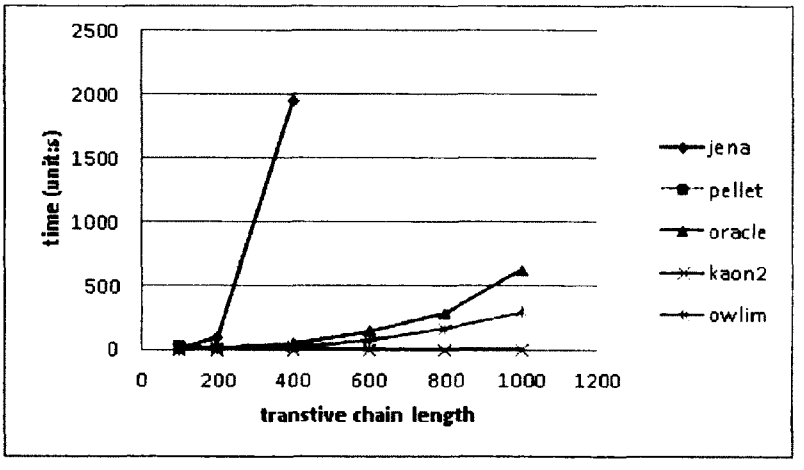


Fig. 1. Setup time for transitive rule

The comparison of different semantic reasoning systems is shown in Table VII.

TABLE VII
COMPARISON OF DIFFERENT SEMANTIC REASONING SYSTEMS

				OWLIM		
	Kaon2	**Pellet**	**Jena**	**SwiftOWLIM**	**BigOWLIM**	**Oracle**
Supported expressive power for reasoning	SHIQ(D)	SROIQ(D)	varies by reasoner	R-entailment, OWL 2 RL		OWL: union, intersection, OWL 2 RL
Reasoning algorithm	Resolution & Datalog	Tableau	Rule-based	Rule-based	Rule-based	N/A
In-memory reasoning	Yes	Yes	Yes	Yes	No	No
Materialization	No	Yes	Yes	Yes	Yes	Yes
Open-source	No	Yes	Yes	No	No	No

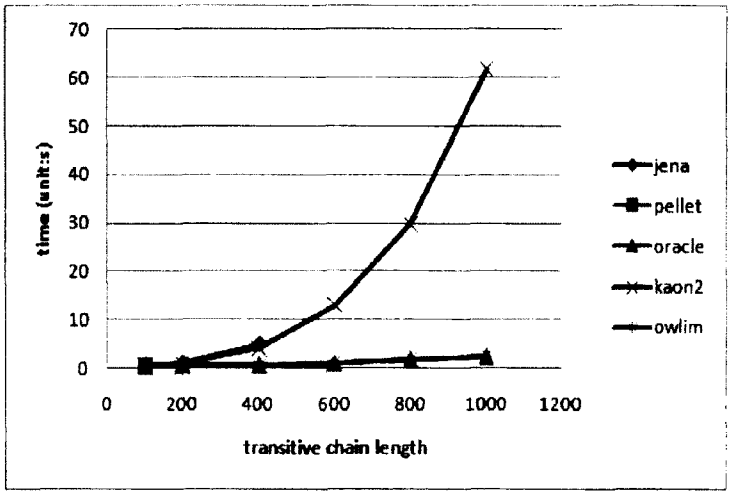


Fig. 2. Query processing time after inference over transitive rule

3.5 Additional Techniques for Scalability

In this section, I explore the additional techniques for improving the scalability of the ontology reasoner. I will present transformation of RDFS rules [115] and OWL Horst fragments [116] for a given ontology such that reasoning will be more efficient. I will

also provide techniques of loading RDF files by property during the reasoning process instead of pre-loading them all.

3.6 Specialized Rule Sets

In order to improve the scalability of the ontology reasoning, I transform RDFS rules and OWL Horst fragments by incorporating the semantics of the ontology into the rules directly.

3.6.1 Description of the Transformation

An example of a rule is:

$(x_1\ p_1\ y_1)\ \ (x_2\ p_2\ y_2)\ (p_1\ p_b^1\ c_1)\ (p_2\ p_b^2\ c_2)$ -> $(x_1\ p_b^1\ c_1)$

where x_1, x_2, y_1, y_2, c_1, c_2, p_1, p_2 are variables, and p_b^1 p_b^2 are known properties.

The first step is to perform Tbox reasoning (reasoning about ontology class and property axioms).

The second step is to specialize the premises that are related to Tbox reasoning, such as $(p_1\ p_b^1\ c_1)\ (p_2\ p_b^2\ c_2)$, which is to find all of the facts in the knowledge base including inferred facts that make $(p_1\ p_b^1\ c_1)\ (p_2\ p_b^2\ c_2)$ true.

All of the matched triples that can make $(p_1\ p_b^1\ c_1)\ (p_2\ p_b^2\ c_2)$ true are:

$(p_1^1\ p_b^1\ c_1^1)\ (p_2^1\ p_b^2\ c_2^1)$
$(p_1^2\ p_b^1\ c_1^2)\ (p_2^2\ p_b^2\ c_2^2)$
.....
$(p_1^k\ p_b^1\ c_1^k)\ (p_2^k\ p_b^2\ c_2^k)$

where $N>=k>=0$ (we define N as the number of triples in the storage).

The third step is to substitute the matched values into the original rule form to

specialize the rule sets and remove the redundant premises.

The rule form turns into:

$(x_1\ p_1^1\ y_1)\ \ (x_2\ p_2^1\ y_2)\ \text{->}\ (x_1\ p_b^1\ c_1^1)$
$(x_1\ p_1^2\ y_1)\ \ (x_2\ p_2^1\ y_2)\ \text{->}\ (x_1\ p_b^1\ c_1^2)$
.....
$(x_1\ p_1^k\ y_1)\ \ (x_2\ p_2^1\ y_2)\ \text{->}\ (x_1\ p_b^1\ c_1^k)$

3.6.2 Proof of Soundness and Completeness of the Specialized Rule Sets

Given the description of our method, our goal is to prove the exact same triples will be inferred when firing the original rules and the specialized rules.

Soundness of specialized rules: The new rule set will be sound iff all of the triples that could be inferred from the specialized rules will satisfy the original rules.

$(p_1^1\text{-}p_1^k)$, $(p_2^1\text{-}p_2^k)$, $(c_1^1\text{-}c_1^k)$ and $(c_2^1\text{-}c_2^k)$ can make $(p_1\ p_b^1\ c_1)\ (p_2\ p_b^2\ c_2)$ true.

All the triples inferred by

$(x_1\ p_1^1\ y_1)\ \ (x_2\ p_2^1\ y_2)\ \text{->}(x_1\ p_b^1\ c_1^1)$
$(x_1\ p_1^2\ y_1)\ \ (x_2\ p_2^2\ y_2)\ \text{->}\ (x_1\ p_b^1\ c_1^2)$
.....
$(x_1\ p_1^k\ y_1\ )\ \ (x_2\ p_2^k\ y_2\ )\ \text{->}\ (x_1\ p_b^1\ c_1^k)$

will make $(x_1\ p_1\ y_1)(x_2\ p_2\ y_2)$ true. Thus

$(x_1\ p_1\ y_1)(x_2\ p_2\ y_2)(p_1\ p_b^1\ c_1)(p_2\ p_b^2\ c_2)$
is true.

Completeness of specialized rule forms: The new rule set will be complete iff all of the triples that could be inferred from the original rules could also be inferred from the specialized rules.

Suppose there is one triple set that makes $(x_1\ p_1\ y_1)\ (x_2\ p_2\ y_2)\ (p_1\ p_b^1$

c_1) (p_2 p_b^2 c_2) true but does not satisfy any of

(x_1 p_1^1 y_1) (x_2 p_2^1 y_2)
(x_1 p_1^2 y_1) (x_2 p_2^2 y_2)
.....
(x_1 p_1^k y_1) (x_2 p_2^k y_2)

Suppose the triple set is (x_1^0 p_1^0 y_1^0) (x_2^0 p_2^0 y_2^0) (p_1^0 p_b^1 c_1^0) (p_2^0 p_b^2 c_2^0).

Thus, (x_1^0 p_1^0 y_1^0) is true, (x_2^0 p_2^0 y_2^0) is true, (p_1^0 p_b^1 c_1^0) is true and (p_2^0 p_b^2 c_2^0) is true. Then (p_1^0 p_b^1 c_1^0) (p_2^0 p_b^2 c_2^0) is definitely true.

Because (p_1^1-p_1^k), (p_2^1-p_2^k), (c_1^1-c_1^k) and (c_2^1-c_2^k) can make (p_1 p_b^1 c_1) (p_2 p_b^2 c_2) true, and these are all we can find, p_1^0 is among (p_1^1-p_1^k) and p_2^0 is among (p_2^1-p_2^k).

So, one of

(x_1 p_1^1 y_1) (x_2 p_2^1 y_2)
(x_1 p_1^2 y_1) (x_2 p_2^2 y_2)
.....
(x_1 p_1^k y_1) (x_2 p_2^k y_2)
would be true.

This contradicts our assumption that one triple set that makes (x_1 p_1 y_1) (x_2 p_2 y_2) (p_1 p_b^1 c_1) (p_2 p_b^2 c_2) true but does not satisfy any of

(x_1 p_1^1 y_1) (x_2 p_2^1 y_2)
(x_1 p_1^2 y_1) (x_2 p_2^2 y_2)
.....
(x_1 p_1^k y_1) (x_2 p_2^k y_2).
Thus, all of the triples that are inferred by the original rules will also be inferred by the specialized rules.

3.6.3 Generalized Rule Form and Examples of Specialized Rules

The following is a generalization of the rule form in Section 3.6.1:

$(x_1\ p_1\ y_1)\ (x_2\ p_2\ y_2)\ldots\ldots(x_m\ p_m\ y_m)\ (a_1\ p_a^1\ b_1)\ (a_2\ p_a^2\ b_2)\ldots\ldots(a_n\ p_a^n\ b_n)$
$(p_i^1\ p_b^1\ c_1)\ (p_i^2\ p_b^2\ c_2)\ldots\ldots(p_i^k\ p_b^k\ c_k)$ -> $(x_j\ p_a^j/p_b^j\ y_j)$

The same proof procedure can be applied to this general form to prove the identity of the original rules to the specialized rules.

Based on the description of the transformation from original rules to specialized rules, I present a sample of generated specialized rules in Table VIII. The prefix defined for namespace is "ub".

TABLE VIII
EXAMPLES OF SPECIALIZED RULES

Original Rules	Specialized Rules
(?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)	(?x ub:worksFor ?y) -> (?x ub:memberOf ?y)
	(?x ub:undergraduateDegreeFrom ?y) -> (?x ub:degreeFrom ?y)
	(?x ub:mastersDegreeFrom ?y) -> (?x ub:degreeFrom ?y)
	(?x ub:doctoralDegreeFrom ?y) -> (?x ub:degreeFrom ?y)
	(?x ub:headOf ?y) -> (?x ub:worksFor ?y)
(?a rdfs:subPropertyOf ?b), (?b rdfs:subPropertyOf ?c) -> (?a rdfs:subPropertyOf ?c)	(?x ub:headOf ?y) -> (?x ub:memberOf ?y)
(?P owl:inverseOf ?Q), (?X ?P ?Y) -> (?Y ?Q ?X)	(?x ub:memberOf ?y) -> (?y ub:member ?x)
	(?x ub:member ?y) -> (?y ub:memberOf ?x)
	(?x ub:hasAlumnus ?y) -> (?y ub:degreeFrom ?x)
	(?x ub:degreeFrom ?y) -> (?y ub:hasAlumnus ?x)
(?x rdfs:subClassOf ?y), (?a rdf:type ?x) -> (?a rdf:type ?y)	(?x rdf:type ub:TechnicalReport) -> (?x rdf:type ub:Article)
	(?x rdf:type ub:University) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:FullProfessor) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:AdministrativeStaff) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:Chair) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:Professor) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:Faculty) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:Manual) -> (?x rdf:type ub:Publication)

TABLE VIII (CONTINUED)

Original Rules	Specialized Rules
(?x rdfs:subClassOf ?y), (?a rdf:type ?x) -> (?a rdf:type ?y)	(?x rdf:type ub:JournalArticle) -> (?x rdf:type ub:Article)
	(?x rdf:type ub:Course) -> (?x rdf:type ub:Work)
	(?x rdf:type ub:UndergraduateStudent) -> (?x rdf:type ub:Student)
	(?x rdf:type ub:Program) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:ConferencePaper) -> (?x rdf:type ub:Article)
	(?x rdf:type ub:SystemsStaff) -> (?x rdf:type ub:AdministrativeStaff)
	(?x rdf:type ub:ResearchGroup) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:Book) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:Specification) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:Software) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:Department) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:Research) -> (?x rdf:type ub:Work)
	(?x rdf:type ub:Dean) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:AssociateProfessor) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:Lecturer) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:ResearchAssistant) -> (?x rdf:type ub:Student)
	(?x rdf:type ub:College) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:PostDoc) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:Institute) -> (?x rdf:type ub:Organization)
	(?x rdf:type ub:Article) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:UnofficialPublication) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:VisitingProfessor) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:GraduateCourse) -> (?x rdf:type ub:Course)
	(?x rdf:type ub:AssistantProfessor) -> (?x rdf:type ub:Professor)
	(?x rdf:type ub:GraduateStudent) -> (?x rdf:type ub:Person)

TABLE VIII (CONTINUED)

Original Rules	Specialized Rules
(?x rdfs:subClassOf ?y), (?a rdf:type ?x) -> (?a rdf:type ?y)	(?x rdf:type ub:ClericalStaff) -> (?x rdf:type ub:AdministrativeStaff)
	(?x rdf:type ub:SystemsStaff) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:ClericalStaff) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:FullProfessor) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:Chair) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:Dean) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:AssociateProfessor) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:VisitingProfessor) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:AssistantProfessor) -> (?x rdf:type ub:Faculty)
	(?x rdf:type ub:TechnicalReport) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:JournalArticle) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:ConferencePaper) -> (?x rdf:type ub:Publication)
	(?x rdf:type ub:FullProfessor) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:Chair) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:Dean) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:AssociateProfessor) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:VisitingProfessor) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:AssistantProfessor) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:Lecturer) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:PostDoc) -> (?x rdf:type ub:Employee)
	(?x rdf:type ub:GraduateCourse) -> (?x rdf:type ub:Work)
(?x ?p ?y), (?p rdfs:range ?c) -> (?y rdf:type ?c)]	(?x ub:affiliatedOrganizationOf ?y) -> (?y rdf:type ub:Organization)
	(?x ub:teacherOf ?y) -> (?y rdf:type ub:Course)
	(?x ub:advisor ?y) -> (?y rdf:type ub:Professor)
	(?x ub:softwareDocumentation ?y) -> (?y rdf:type ub:Publication)
	(?x ub:teachingAssistantOf ?y) -> (?y rdf:type ub:Course)

TABLE VIII (CONTINUED)

Original Rules	Specialized Rules
(?x ?p ?y), (?p rdfs:range ?c) -> (?y rdf:type ?c)]	(?x ub:member ?y) -> (?y rdf:type ub:Person)
	(?x ub:researchProject ?y) -> (?y rdf:type ub:Research)
	(?x ub:affiliateOf ?y) -> (?y rdf:type ub:Person)
	(?x ub:orgPublication ?y) -> (?y rdf:type ub:Publication)
	(?x ub:mastersDegreeFrom ?y) -> (?y rdf:type ub:University)
	(?x ub:degreeFrom ?y) -> (?y rdf:type ub:University)
	(?x ub:hasAlumnus ?y) -> (?y rdf:type ub:Person)
	(?x ub:subOrganizationOf ?y) -> (?y rdf:type ub:Organization)
	(?x ub:publicationResearch ?y) -> (?y rdf:type ub:Research)
	(?x ub:publicationAuthor ?y) -> (?y rdf:type ub:Person)
	(?x ub:undergraduateDegreeFrom ?y) -> (?y rdf:type ub:University)
	(?x ub:doctoralDegreeFrom ?y) -> (?y rdf:type ub:University)
	(?x ub:listedCourse ?y) -> (?y rdf:type ub:Course)
(?x ?p ?y), (?p rdfs:domain ?c) -> (?x rdf:type ?c)	(?x ub:publicationAuthor ?y) -> (?x rdf:type ub:Publication)
	(?x ub:undergraduateDegreeFrom ?y) -> (?x rdf:type ub:Person)
	(?x ub:telephone ?y) -> (?x rdf:type ub:Person)
	(?x ub:doctoralDegreeFrom ?y) -> (?x rdf:type ub:Person)
	(?x ub:mastersDegreeFrom ?y) -> (?x rdf:type ub:Person)
	(?x ub:emailAddress ?y) -> (?x rdf:type ub:Person)
	(?x ub:advisor ?y) -> (?x rdf:type ub:Person)
	(?x ub:age ?y) -> (?x rdf:type ub:Person)
	(?x ub:softwareVersion ?y) -> (?x rdf:type ub:Software)
	(?x ub:publicationResearch ?y) -> (?x rdf:type ub:Publication)
	(?x ub:affiliateOf ?y) -> (?x rdf:type ub:Organization)
	(?x ub:title ?y) -> (?x rdf:type ub:Person)
	(?x ub:affiliatedOrganizationOf ?y) -> (?x rdf:type ub:Organization)
	(?x ub:orgPublication ?y) -> (?x rdf:type ub:Organization)

TABLE VIII (CONTINUED)

Original Rules	Specialized Rules
(?x ?p ?y), (?p rdfs:domain ?c) -> (?x rdf:type ?c)	(?x ub:teacherOf ?y) -> (?x rdf:type ub:Faculty)
	(?x ub:degreeFrom ?y) -> (?x rdf:type ub:Person)
	(?x ub:listedCourse ?y) -> (?x rdf:type ub:Schedule)
	(?x ub:member ?y) -> (?x rdf:type ub:Organization)
	(?x ub:hasAlumnus ?y) -> (?x rdf:type ub:University)
	(?x ub:tenured ?y) -> (?x rdf:type ub:Professor)
	(?x ub:researchProject ?y) -> (?x rdf:type ub:ResearchGroup)
	(?x ub:subOrganizationOf ?y) -> (?x rdf:type ub:Organization)
	(?x ub:teachingAssistantOf ?y) -> (?x rdf:type ub:TeachingAssistant)
	(?x ub:publicationDate ?y) -> (?x rdf:type ub:Publication)
(?P rdf:type owl:TransitiveProperty), (?A ?P ?B), (?B ?P ?C) -> (?A ?P ?C)	(?x ub:subOrganizationOf ?y) (?y ub:subOrganizationOf ?z) -> (?x ub:subOrganizationOf ?z)

3.6.4 Preliminary Evaluation of Jena Using the Specialized Rules

To present the effectiveness of the specialized rules, I evaluate the 14 queries provided by the LUBM using Jena on top of two rule sets. One rule set contains the original RDFS rules and OWL Horst fragments. The other rule set contains the specialized rules that capture the semantics from the LUBM ontology. The experiments are run in memory. So, considering the limits of an in-memory test, I only employ two small datasets, LUBM (01) (one department of a university) and LUBM (1) (one university), of size 8820 and 100844 respectively, for the experiments. Jena supports three ways of inferencing: forward-chaining, backward-chaining and a hybrid of these two methods. In this experiment, I select the hybrid mode of Jena, since it is the default mode and considered to be a performance tradeoff [129]. The evaluation results are

presented in Table IX.

TABLE IX
EVALUATION OF JENA USING THE SPECIALIZED RULES AND ORIGINAL RULES

	Query Response time (ms) Specialized rules	**Query Response time (ms) Original rules**	**Query Response time (ms) Specialized rules**	**Query Response time (ms) Original rules**
	LUBM(01)		*LUBM1*	
Query1	757	5394	1381	406212
Query2	1552	1659	119176	243130
Query3	4	5	20	28
Query4	11	11	16	74583
Query5	67	171	105	954703
Query6	45	47	412	432
Query7	227	20375	27310	Out of Memory Error
Query8	101	118	838	1465351
Query9	7182	94411	15197259	>1 hour
Query10	3	4	31	1507150
Query11	1	3	5	995
Query12	1	18	2	8465
Query13	2	12	16	198
Query14	23	24	102	286

As Table IX shows, for LUBM (01), Jena with the specialized rules has much better performance for query 1, query 5, query 7 and query 9. For the other queries, Jena with the specialized rules has similar performance as the original rules. For LUBM (1), Jena with specialized rules has much better performance for most of the 14 queries except for query 3, query 6, query 13 and query 14. Jena with the original rules is not able to return answers in one hour (query 9) or terminates abnormally due to an "Out of Memory Error" (query 7). In general, Jena with the specialized rules outperforms Jena with the original rules for both datasets.

All the results returned in the above experiments are sound and complete. In the

above experiment, I have not taken the processing time for the transformation from the original rules into the specialized rules into account. This processing time will not affect the dramatic difference between the performance of the specialized rules and original rules especially significantly, particularly, as the size of the dataset increases. There are two reasons. First, the transformation is a one-time job if the ontology is stable, so the processing time can be distributed into each query response. Second, the transformation normally can be completed within seconds for LUBM as tested.

3.7 Discussions

One of the promises of the evolving Semantic Web is that it will enable systems that can handle qualitative queries such as "good PhD advisors in data mining". We have explored in this chapter the feasibility of developing such a system using ontologies and reasoning system that can handle customized rules such as "validated co-author".

When looking at more realistic models (ScienceWeb) than provided by LUBM serious issues arise when the size approaches millions of triplets. For the most part, OWLIM and Oracle offer the best scalability for the kinds of datasets anticipated for ScienceWeb.

This scalability comes in part from heavy front-loading of the inferencing costs by pre-computing the entailed relationships at set-up time. This, in turn, has negative implications for evolving systems. One can tolerate high one-time costs that are needed to set up a reasoning system for a particular ontology and an instance set. If, however, the ontologies or rule sets evolve at a significant rate, then repeated incurrence of these high setup cost is likely to be prohibitively expensive.

The times we obtained for some queries indicate that real-time queries over large

triplet spaces will have to be limited in their scope unless one gives up on the answers being returned in real time. The question as to how we can specify what can be asked within a real-time system is an open one.

CHAPTER 4

OPTIMIZED QUERY-ANSWERING ALGORITHM

Interposing a backward chaining reasoner between a knowledge base and a query manager yields an architecture that can support reasoning in the face of frequent changes. However, such an interposition of the reasoning introduces uncertainty regarding the size and effort measurements typically exploited during query optimization. In this section, I present an algorithm for dynamic query optimization in such an architecture and experimental results confirming its effectiveness.

4.1 Dynamic Query Optimization

A query is typically posed as the conjunction of a number of clauses. In a traditional data base, each clause may denote a distinct probe of the data base contents. Easily accessible information about the anticipated size and other characteristics of such probes can be used to facilitate query optimization. The interposition of a reasoner between the query handler and the underlying knowledge base means that not all clauses will be resolved by direct access to the knowledge base. Some will be handed off to the reasoner, and the size and other characteristics of the responses to such clauses cannot be easily predicted in advance. If the reasoner is associated with an ontology, however, it may be possible to relieve this problem by exploiting knowledge about the data types introduced in the ontology.

In this section, I describe an algorithm for resolving such queries using dynamic optimization based, in part, upon summary information associated with the ontology. I

begin with the definitions of the fundamental data types that we will be manipulating. Then I discuss the algorithm for answering a query. A running example is provided to make the process more understandable.

We model the knowledge base as a collection of triples. A triple is a 3-tuple (x, p, y) where x, p, and y are URIs or constants and where p is generally interpreted as the identifier of a property or predicate relating x and y. For example, a knowledge base might contains triples

```
(Jones, majorsIn, CS), (Smith, majorsIn, CS),
(Doe, majorsIn, Math), (Jones, registeredIn, Calculus1), (Doe,
registeredIn, Calculus1).
```

A QueryPattern is a triple in which any of the three components can be occupied by references to one of a pool of entities considered to be variables. In our examples, we will denote variables with a leading '?'. For example, a query pattern denoting the idea "Which students are registered in Calculus1?" could be shown as

```
(?Student,registeredIn,Calculus1).
```

A query is a request for information about the contents of the knowledge base. The input to a query is modeled as a sequence of QueryPatterns. For example, a query "What are the majors of students registered in Calculus1?" could be represented as the sequence of two query patterns

```
[(?Student,registeredIn,Calculus1),
  (?Student, majorsIn, ?Major)].
```

The output from a query will be a QueryResponse. A QueryResponse is a set of functions mapping variables to values in which all elements (functions) in the set share a common domain (i.e., map the same variables onto values). Mappings from the same variables to values can be also referred to as variable bindings. For example, the QueryResponse of query pattern (?Student, majorsIn, ?Major) could be the set

```
{{?Student => Jones, ?Major=>CS},
  {?Student => Smith, ?Major=>CS },
   {?Student => Doe, ?Major=> Math }}.
```

The SolutionSpace is an intermediate state of the solution during query processing, consisting of a sequence of (preliminary) QueryResponses, each describing a unique domain. For example, the SolutionSpace of the query "What are the majors of students registered in Calculus1?" that could be represented as the sequence of two query patterns as described above could first contain two QueryResponses:

```
[{{?Student => Jones, ?Major=>CS},
   {?Student => Smith, ?Major=>CS },
    {?Student => Doe, ?Major=> Math }},
  {{?Student => Jones},{?Student => Doe }}]
```

Each QueryResponse is considered to express a constraint upon the universe of possible solutions, with the actual solution being intersection of the constrained spaces. An equivalent SolutionSpace is therefore:

```
[{{?Student => Jones, ?Major=>CS},
  {?Major => Math, ?Student =>Doe}}],
```

Part of the goal of our algorithm is to eventually reduce the SolutionSpace to a single QueryResponse like this last one.

Fig. 8 describes the top-level algorithm for answering a query. A query is answered by a process of progressively restricting the SolutionSpace by adding variable bindings (in the form of QueryResponses). The initial space with no bindings ❶ represents a completely unconstrained SolutionSpace. The input query consists of a sequence of query patterns.

We repeatedly estimate the response size for the remaining query patterns ❷, and choose the most restrictive pattern ❸ to be considered next. We solve the chosen pattern by backward chaining ❹, and then merge the variable bindings obtained from backward

chaining into the SolutionSpace ❺ via the restrictTo function, which performs a (possibly deferred) join as described later in this section.

```
QueryResponse answerAQuery(query: Query)
{
   // Set up initial SolutionSpace
   SolutionSpace solutionSpace = empty; ❶

   // Repeatedly reduce SolutionSpace by applying
   //   the most restrictive pattern
   while (unexplored patterns remain in the query) {

     computeEstimatesOfReponseSize (unexplored patterns); ❷
     QueryPattern p = unexplored pattern with smallest estimate; ❸

       // Restrict SolutionSpace via exploration of p
      QueryResponse answerToP = BackwardChain(p); ❹
      solutionSpace.restrictTo(answerToP); ❺
    }
    return solutionSpace.finalJoin();❻
}
```

Fig. 3. Answering a Query

When all query patterns have been processed, if the SolutionSpace has not been reduced to a single QueryResponse, we perform a final join of these variable bindings into single one variable binding that contains all the variables involved in all the query patterns ❻. The finalJoin function is described in detail later in this section.

The estimation of response sizes in ❷ can be carried out by a combination of 1) exploiting the fact that each pattern represents that application of a predicate with known domain and range types. If these positions in the triple are occupied by variables, we can check to see if the variable is already bound in our SolutionSpace and to how many values it is bound. If it is unbound, we can estimate the size of the domain (or range) type, 2) accumulating statistics on typical response sizes for previously encountered

patterns involving that predicate. The effective mixture of these sources of information is a subject for future work.

For example, suppose there are 10,000 students, 500 courses, 50 faculty members and 10 departments in the knowledgebase. For the query pattern (`?S takesCourse ?C`), the domain of `takesCourse` is Student, while the range of `takesCourse` is Course. An estimate of the numbers of triples matching the pattern (`?S takesCourse ?C`) might be 100,000 if the average number of courses a student has taken is ten, although the number of possibilities is 500,000.

By using a greedy ordering ❸ of the patterns within a query, we hope to reduce the average size of the `SolutionSpaces`. For example, suppose that we were interested in listing all cases where any student took multiple courses from a specific faculty member. We can represent this query as the sequence of the following patterns, shown in Table X with their estimated result size (the sizes are based on one of our LUBM benchmark prototypes).

TABLE X
QUERYPATTENS AND ESTIMATED RESULT SIZE

Clause #	QueryPattern	QueryResponse
1	?S1 takesCourse ?C1	$\{(?S1=>s_i,?C1=>c_i)\}_{i=1..100,000}$
2	?S1 takesCourse ?C2	$\{(?S1=>s_j, ?C2=>c_j)\}_{j=1..100,000}$
3	?C1 taughtBy fac1	$\{(?C1=>c_j)\}_{j=1..3}$
4	?C2 taughtBy fac1	$\{(?C2=>c_j)\}_{j=1..3}$

To illustrate the effect of the greedy ordering, let us assume first that the patterns are processed in the order given. A trace of the `answerAQuery` algorithm, showing one row for each iteration of the main loop would be in Table XI.

TABLE XI
TRACE OF JOIN OF CLAUSES IN THE ORDER GIVEN

Clause Being Joined	Resulting SolutionSpace
(initial)	[ ]
1	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i)\}_{i=1..100,000}]$
2	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i, ?C2 \Rightarrow c_i)\}_{i=1..1,000,000}]$ (based on an average of 10 courses per student)
3	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i, ?C2 \Rightarrow c_i)\}_{i=1..900}]$ (Joining this clause discards courses taught by other faculty.)
4	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i, ?C2 \Rightarrow c_i)\}_{i=1..60}]$

The worst case in terms of storage size and in terms of the size of the sets being joined was at the join of clause 2, when the join of two sets of size 100,000 yielded 1,000,000 tuples, as shown in above Table XI.

Now, consider the effect of applying the same patterns in ascending order of estimated size, shown in above Table XII.

TABLE XII
TRACE OF JOIN OF CLAUSES IN ASCENDING ORDER OF ESTIMATED SIZE

Clause Being Joined	Resulting SolutionSpace
(initial)	[ ]
3	$[[\{(?C1 \Rightarrow c_i)\}_{i=1..3}]$
4	$[\{(?C1 \Rightarrow c_i, ?C2 \Rightarrow c_j)\}_{i=1..3, j=1..3}]$
1	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i, ?C2 \Rightarrow c'_i)\}_{i=1..270}]$
2	$[\{(?S1 \Rightarrow s_i, ?C1 \Rightarrow c_i, ?C2 \Rightarrow c_i)\}_{i=1..60}]$

The worst case in terms of storage size and in terms of the size of the sets being joined was at the final addition of clause 2, when a set of size 100,000 was joined with a set of 270. The reduction in space requirements and in time required to perform the join would be about an order of magnitude.

The output from the backward chaining reasoner will be a query response. These

must be merged into the current SolutionSpace as a set of additional restrictions. Fig. 9 shows how this is done.

```
void SolutionSpace::restrictTo (QueryResponse newbinding)
{
   for each element oldBinding in solutionSpace
   {
      if (newbinding shares variables with oldbinding){ ❶
          bool merged = join(newBinding,oldBinding, false);❷
          if (merged) {
              remove oldBinding fromsolutionSpace;
          }
      }
   }
   add newBinding to solutionSpace;
}
```

Fig. 4. Restricting a SolutionSpace

Each binding already in the SolutionSpace ❶ that shares at least one variable with the new binding ❷ is applied to the new binding, updating the new binding so that its domain is the union of the sets of variables in the old and new bindings and the specific functions represent the constrained cross-product (join) of the two. Any such old bindings so joined to the new one can then be discarded.

The join function at ❷ returns the joined QueryResponse as an update of its first parameter. The join operation is carried out as a hash join [168] with an average complexity O(n1+n2+m) where the ni are the number of tuples in the two input sets and m is the number of tuples in the joined output.

The third (boolean) parameter of the join call indicates whether the join is forced (true) or optional (false), and the boolean return value indicates whether an optional join was actually carried out. Our intent is to experiment in future versions with a dynamic

decision to defer optional joins if a partial calculation of the join reveals that the output will far exceed the size of the inputs, in hopes that a later query clause may significantly restrict the tuples that need to participate in this join.

As noted earlier, our interpretation of the `SolutionSpace` is that it denotes a set of potential bindings to variables, represented as the join of an arbitrary number of `QueryResponses`. The actual computation of the join can be deferred, either because of a dynamic size-based criterion as just described, or because of the requirement at ❶ that joins be carried out immediately only if the input `QueryResponses` share at least one variable. In the absence of any such sharing, a join would always result in an output size as long as the products of its input sizes. Deferring such joins can help reduce the size of the `SolutionSpace` and, as a consequence, the cost of subsequent joins.

For example, suppose that we were processing a query to determine which mathematics courses are taken by computer science majors, represented as the sequence of the following `QueryPatterns`, shown with their estimated sizes in Table XIII.

TABLE XIII
QUERYPATTERNS AND THEIR ESTIMATED SIZES

Clause	QueryPattern	QueryResponse
1	(?S1 takesCourse ?C1)	$\{(?S1 \Rightarrow s_j, ?C1 \Rightarrow c_j)\}_{j=1..100,000}$
2	(?S1 memberOf CSDept)	$\{(?S1 \Rightarrow s_j)\}_{j=1..1,000}$
3	(?C1 taughtby ?F1)	$\{(?C1 \Rightarrow c_j, ?F1 \Rightarrow f_j)\}_{j=1..1,500}$
4	(?F1 worksFor MathDept)	$\{(?F1 \Rightarrow f_i)\}_{i=1..50}$

To illustrate the effect of deferring joins on responses that do not share variables, even with the greedy ordering discussed earlier, suppose, first, that we perform all joins immediately. Assuming the greedy ordering that we have already advocated, the trace of

the answerAQuery algorithm would be in Table XIV.

TABLE XIV
TRACE OF JOIN OF CLAUSES IN ASCENDING ORDER OF ESTIMATED SIZE

Clause Being Joined	Resulting SolutionSpace
(initial)	[]
4	$[\{(?F1=>f_i)\}_{i=1..50}]$
2	$[\{(?F1=>f_i, ?S1=>s_i)\}_{i=1..50,000}]$
3	$[\{(?F1=>f_i, ?S1=>s_i, ?C1=>c_i)\}_{i=1..150,000}]$
1	$[\{(?F1=>f_i, ?S1=>s_i, ?C1=>c_i)\}_{i=1..1,000}]$

In the prototype from which this example is taken, the Math department teaches 150 different courses and there are 1,000 students in the CS Dept. Consequently, the merge of clause 3 (1,500 tuples) with the SolutionSpace then containing 50,000 tuples yields considerably fewer tuples than the product of the two input sizes.

The worst step in this trace was the final join, between sets of size 100,000 and 150,000.

But consider that the join of clause 2 in that trace was between sets that shared no variables. If we defer such joins, then the first SolutionSpace would be retained "as is". The resulting trace would be shown in Table XV.

TABLE XV
TRACE OF JOIN OF CLAUSES WITH DEFERRING

Clause Being Joined	Resulting SolutionSpace
(initial)	[]
4	$[\{(?F1=>f_i)\}_{i=1..50}]$
2	$[\{(?F1=>f_i)\}_{i=1..50}, \{(?S1=>s_j)\}_{j=1..1,000}]$
3	$[\{(?F1=>f_i, ?C1=>c_i)\}_{i=1..150}, \{(?S1=>s_j)\}_{j=1..1,000}]$
1	$[\{(?F1=>f_i, ?S1=>s_i, ?C1=>c_i)\}_{i=1..1,000}]$

The subsequent addition of clause 3 results in an immediate join with only one of the responses in the solution space. The response involving ?S1 remains deferred, as it shares no variables with the remaining clauses in the SolutionSpace.

The worst join performed would have been between sets of size 100,000 and 150, a considerable improvement over the non-deferred case.

```
QueryResponseSolutionSpace::finalJoin ()
{
   sort the bindings in this solution space into
     descending order by number of tuples;  ❶

   QueryResponse result = first of the sorted bindings;
   for each remaining binding b in solutionSpace {
     join (result, b, true);  ❷
   }
   return result;
}
```

Fig. 5. Final Join

When all clauses of the original query have been processed (Fig. 8 ❻), we may have deferred several joins because they involved unrelated variables or because they appeared to lead to a combinatorial explosion on their first attempt. The finalJoin function shown in Fig. 10 is tasked with reducing the internal SolutionSpace to a single QueryResponse, carrying out any join operations that were deferred by the earlier restrictTo calls. In many ways, finalJoin is a recap of the answerAQuery and restrictTo functions, with two important differences:

- Although we still employ a greedy ordering ❶ to reduce the join sizes, there is no need for estimated sizes because the actual sizes of the input QueryResponses are known.

- There is no longer an option to defer joins between `QueryResponses` that share no variables. All joins must be performed in this final stage❷ and so the "forced" parameter to the optional join function is set to true.

4.2 Evaluation of the Query-answering Algorithm

In this section I compare our `answerAQuery` algorithm of Fig. 8 against an existing system, Jena, that also answers queries via a combination of an in-memory backward chaining reasoner with basic knowledge base retrievals.

The comparison was carried out using two LUBM benchmarks representing a knowledge base describing a single university and one with 10 universities. Prior to the application of any reasoning, these benchmarks contained 100,839 and 1,272,871 triples, respectively.

I evaluated these using the set of 14 queries taken from LUBM [161]. These queries involve properties associated with the LUBM university-world ontology, with none of the custom properties/rules whose support is actually our end goal (as discussed in [5]). Answering these queries requires, in general, reasoning over rules associated with both RDFS and OWL semantics, though some queries can be answered purely on the basis of the RDFS rules.

Table XVI compares our algorithm to the Jena system using a pure backward chaining reasoner. Jena's system cannot process all of the rules in the OWL semantics rule set, and was therefore run with a simpler rule set describing only the RDFS semantics. This discrepancy accounts for the differences in result size (# of tuples) for several queries. Result sizes in the table are expressed as the number of tuples returned by the query and response times are given in seconds. An entry of n/a means that the query

processing had not completed (after 1 hour).

Despite employing the larger and more complicated rule set, our algorithm generally ran faster than Jena, sometimes by multiple orders of magnitude. The exceptions to this behavior are limited to queries with very small result set sizes or queries 10-13, which rely upon OWL semantics and so could not be answered correctly by Jena. In two queries (2 and 9), Jena timed out.

TABLE XVI
COMPARISON AGAINST JENA WITH BACKWARD CHAINING

LUBM:	1 University, 100,839 triples				10 Universities, 1,272,871 triples			
	answerAQuery		Jena Backwd		answerAQuery		Jena Backwd	
	response time (s)	result size (tuple)	response time (s)	result size (tuple)	response time (s)	result size (tuple)	response time (s)	result size (tuple)
Query1	0.26	4	0.32	4	25.	4	0.86	4
Query2	0.49	0	130	0	0.69	0	n/a	n/a
Query3	0.056	6	0.038	6	1.5	6	1.5	6
Query4	0.47	34	0.021	34	0.034	34	0.41	34
Query5	0.033	719	0.19	678	1.1	719	1.0	678
Query6	0.18	7,790	0.49	6,463	0.023	99,566	3.2	82,507
Query7	0.19	67	45	61	1.7	67	8100	61
Query8	0.54	7,790	0.91	6463	2.2	7,790	52	6,463
Query9	0.25	208	n/a	n/a	2.7	2,540	n/a	n/a
Query10	0.14	4	0.54	0	2.4	4	1.4	0
Query11	0.19	224	0.011	0	1.7	224	0.032	0
Query12	0.22	15	0.0020	0	0.19	15	0.016	0
Query13	0.028	1	0.37	0	0.34	33	0.89	0
Query14	0.024	5,916	0.58	5,916	0.026	75,547	2.6	75,547

Jena also has a hybrid mode that combines backward chaining with some forward-style materialization. Table XVII shows a comparison of our algorithm with a pure backward chaining reasoner against the Jena hybrid mode [4]. Again, an n/a entry indicates that the query processing had not completed within an hour, except in one case (query 8 in the 10 Universities benchmark) in which Jena failed due to exhausted

memory space.

The times here tend to be someone closer, but the Jena system has even more difficulties returning any answer at all when working with the larger benchmark. Given that the difference between this and the prior table is that, in this case, some rules have already been materialized by Jena to yield, presumably, longer lists of tuples, steps taken to avoid possible combinatorial explosion in the resulting joins would be increasingly critical.

TABLE XVII
COMPARISON AGAINST JENA WITH HYBRID REASONER

LUBM	1 University, 100,839 triples				10 Universities, 1,272,871 triples			
	answerAQuery		Jena Hybrid		answerAQuery		Jena Hybrid	
	response time	result size	response time	result size	response time	result size	response time	result size
Query1	0.26	4	0.37	4	25.	4	0.93	4
Query2	0.49	0	1,400	0	0.69	0	n/a	n/a
Query3	0.056	6	0.050	6	1.5	6	1.5	6
Query4	0.47	34	0.025	34	0.034	34	0.55	34
Query5	0.033	719	0.029	719	1.1	719	2.7	719
Query6	0.18	7,790	0.43	6,463	0.023	99,566	3.7	82,507
Query7	0.19	67	38	61	1.7	67	n/a	n/a
Query8	0.54	7,790	2.3	6,463	2.2	7,790	n/a	n/a
Query9	0.25	208	n/a	n/a	2.7	2,540	n/a	n/a
Query10	0.14	4	0.62	0	2.4	4	1.6	0
Query11	0.19	224	0.0010	0	1.7	224	0.08	0
Query12	0.22	15	0.0010	0	0.19	15	0.016	0
Query13	0.028	1	0.62	0	0.34	33	1.2	0
Query14	0.024	5,916	0.72	5,916	0.026	75,547	2.5	75,547

CHAPTER 5

OPTIMIZED BACKWARD CHAINING ALGORITHM

In this chapter, I will introduce new optimization techniques to the backward-chaining. I will show that these techniques together with the query-optimization reported in Chapter 4, will allow us to outperform forward-chaining reasoners in scenarios where the knowledge base is subject to frequent change. Finally, I will analyze the impact of these techniques on a large knowledge base that requires external storage.

5.1 Issues

When the knowledge base is small and dynamic, backward chaining is suitable for ontology reasoning. However, as the size of the knowledge base increases, standard backward chaining [8] do not scale well for ontology reasoning. In this section, I discuss issues that most standard backward chaining methods for ontology reasoning have.

5.1.1 Guaranteed Termination

Backward chaining is usually implemented by employing a depth-first search strategy. Unless methods are used to prevent it, the depth-first search could go into an infinite loop. For example, in the rule set we have used so far, we have rules that involve each other when proving their heads:

```
rule1: (?P owl:inverseOf ?Q) -> (?Q owl:inverseOf ?P)
rule2:(?P owl:inverseOf ?Q), (?X ?P ?Y) -> (?Y ?Q ?X)
```

In order to prove body clause `?P owl:inverseOf ?Q` in rule1, we need to prove the body of rule2 first, because the head of rule2 matches body clause `?P`

owl:inverseOf ?Q. In order to prove the first body clause ?P owl:inverseOf ?Q in rule2, we also need to prove the body clause ?P owl: inverseOf ?Q in rule1, because the head of rule1 matches body clause ?P owl:inverseOf ?Q.

Even in cases where depth-first search terminates, the performance may suffer due to time spent exploring, in depth, branches that ultimately do not lead to a proof.

5.1.2 The owl:sameAs Problem

The built-in OWL property owl:sameAs links two equivalent individuals. An owl:sameAs triple indicates that two linked individuals have the same "identity". [169] An example of a rule in the OWL-Horst rule set that involves the owl:sameAs relations is the rule: "(?x owl:sameAs ?y) (?x ?p ?z) -> (?y ?p ?z)". Consider a triple, which has m owl:sameAs equivalents of its subject, n owl:sameAs equivalents of its predicate, and k owl:sameAs equivalents of its object, Then m*n*k triples would be derivable from that triple.

Reasoning with the owl:sameAs relation can result in a multiplication of the number of instances of variables during backward-chaining and expanded patterns in the result. As long as that triple is in the result set, all of its equivalents would be in the result set as well. This adds cost to the reasoning process in both time and space.

5.2 The Algorithm

The purpose of this algorithm is to generate a query response for a given query pattern based on a specific rule set. I use the following terminology.

The main algorithm calls the function BackwardChaining which finds a set of triples that can be unified with pattern with bindings in varList, any bindings to

variables appearing in `headClause` from the head of an applied rule, `bodylist` and level that are reserved for solving the recursive problem. Given a Goal and corresponding matched triples, a `QueryResponse` is created and returned in the end.

The optimized `BackwardChaining` algorithm, described in Fig. 11, is based on conventional backward chaining algorithms [8].The `solutionList` is a partial list of solutions already found for a goal. For a goal that has already been resolved, we simply get the results from `solutionList` ❶. For a goal that has not been resolved yet, we will seek a resolution by applying the rules ❷. We initially search in the knowledge base to find triples that match the goal (triples in which the subject, predicate and object are compatible with the query pattern) ❸. Then, we find rules with heads that match the input pattern ❹. For each such rule we attempt to prove it by proving the body clauses (new goals) subject to bindings from already-resolved goals from the same body ❺. The process of proving one rule is explained below. The method of "OLDT" [170] is adopted to solve the non-termination issue I mentioned in Section 5.3.3. Finally, we apply any "same as" relations to `candidateTriples` to solve the `owl:sameAs` problem❻. During this process of "`SameAsTripleSearch`", we add all equivalent triples to the existing results to produce complete results.

Fig. 12 shows how to prove one rule, which is a step in Fig. 11. The heart of the algorithm is the loop through the clauses of a rule body, attempting to prove each clause. Some form of selection function is implied that selects the next unproven clause for consideration on each iteration. Traditionally, this would be left-to-right as the clauses are written in the rule. Instead, we order the body clauses by the number of free variables. The rationale for this ordering will be discussed in the following Section 5.3.1.

```
BackwardChaining(pattern,headClause,bodylist,level,varList)
{
   if (pattern not in solutionList){  ❷
      candidateTriples+= matches to pattern that are found in
        knowledge base;  ❸
  solutionList+= mapping from pattern to candidateTriples;
  relatedRules = all rules from  ruleList where the head matches
     the pattern;   ❹
  realizedRules = all the rules in relatedRules with variables
    substituted with values from the pattern ;
  backupvarList = back up clone of varList;
  for (each oneRule in realizedRules){ ❺
     if(attemptToProveRule(oneRule, varList, level)){
        resultList= unify(headClause, varList);
        candidateTriples+= resultList;
     }
     oldCandidateTriples = triples in mappings from solutionList
        such that headClause matches goal;
     if ( oldCandidateTriples not contain candidateTriples){
        update solutionList with candidateTriples;
        if(UpdateafterUnificationofHead(headClause, resultList))
        {
            newCandidateTriples = triples in mappings from
               solutionList such that headClause matches goal;
            candidateTriples+= newCandidateTriples;
        }
     }
   }
}
    else /* if (solutionList.contains(pattern)) */ ❶
{
  candidateTriples+= triples in mappings from solutionList
     such that pattern matches goal;
  Add reasoning context, including head and bodyRest to lookupList;
}
SameAsTripleSearch(candidateTriples) ❻;
return candidateTriples;
}
```

Fig. 6. Process of BackwardChaining

The process of proving one goal (a body clause from a rule) is given in Fig. 13. Before we prove the body clauses (new goals) in each rule, the value of a calculated

dynamic threshold decides whether we perform the substitution or not. We substitute the free variables in the body clause with bindings from previously resolved goals from the same body. The step helps to improve the reasoning efficiency in terms of response time and scalability and will be discussed in Section 5.3.2. We call the `BackwardChaining` function to find a set of triples that can be unified with body clause (new goal) with substituted variables. Bindings will also be updated gradually following the proof of body clauses.

```
attemptToProveRule(oneRule,varList,level) {
   body = rule body of oneRule;
   sort body by ascending number of free variables;
   head = rule head of oneRule;
   for (each bodyClause in body)
   {
      canBeProven = attemptToProveBodyClause (
          bodyClause, body, head, varList, level);
       if (!canBeProven) break;
   }
   return canBeProven;
}
```

Fig. 7. Process of proving one rule

5.3 Optimization Details and Discussion

There are four optimizations that have been introduced in the algorithm for backward chaining. These optimizations are: 1) the implementation of the selection function, which implements the ordering of the body clauses in one rule by the number of free variables, 2) the upgraded substitute function, which implements the substitution of the free variables in the body clauses in one rule based on calculating a threshold that

switches resolution methods, 3) the application of OLDT and 4) solving of the "owl:sameAs" problem. Of these, optimization 1 is an adaptation of techniques employed in other reasoning contexts [171, 172] and optimizations 3 and 4 have appeared in [169, 170] whereas technique 2 is new. I will describe the implementation details of these optimizations below. A preliminary evaluation of these techniques is reported in Section 5.4. More extensive evaluations are reported in Section 5.5 and 5.6.

```
attemptToProveBodyClause(goal, body, head, varList, level)
{
  canBeProven = true;
  dthreshold = Calculate dynamic threshold;
  patternList = get unified patterns by replacing variables in bodyClause
    from varList for current level with calculated dthreshold;
  for(each unifiedPattern in patternList ) {
     if(!unifiedPattern.isGround()) {
         bodyRest = unprocessedPartOf(body, goal);
         triplesFromResolution+= BackwardChaining( unifiedPattern, head,
           bodyRest, level+1, varList);
     }
     else if(unifiedPattern.isGround()) {
        if (knowledgeBase contains unifiedPattern){
           triplesFromResolution+= unifiedPattern;
        }
     }
  }
  if(triplesFromResolution.size()>0) {
     update_varList with varList, triplesFromResolution, goal, and level;
     if (varList==null) {
        canBeProven = false;
     }
  }
  else{
        canBeProven = false;
  }
  return canBeProven;
}
```

Fig. 8. Process of proving one goal

5.3.1 Ordered Selection Function

SLD resolution (Selective Linear Definite clause resolution) [173] is a variant of linear resolution that is complete for Horn clauses and is commonly used in Prolog-like systems. It is also called "top-down" or "goal-directed". Rule based ontology reasoners involving backward chaining are mostly based on SLD Resolution [173]. An example is QueryPIE [165] which performs backward chaining reasoning on large RDF databases with a hybrid algorithm and pre-computation. A second example is Jena [98] with a backward chaining engine that evaluates rules in a top-to-bottom, left-to-right order, as in SLD resolution. A third example is IRIS [174] which is a Datalog reasoner supporting top-down strategies like SLDNF (Selective Linear Definite-clause with Negation as Failure) with optimizations.

The selection function [175] in SLD resolution chooses which goal to prove next which impacts the size of the search space and the efficiency of evaluation [176]. In the simplest case, the selection function can be specified by the order in which literals are written (left-to-right order), such as Prolog. Several reordering techniques can be found in related literature. In Inductive Logic Programming (ILP), two selection functions for SLD resolution have been implemented [172]: Smallest Predicate Domain (SPD-resolution for simplicity) and Smallest Variable Domain (SVD-resolution). SPD selects the literal with the fewest number of solutions at each moment first. SVD binds the variable with the smallest domain with one of its possible values. Reordering has also been applied in CLP program[171] and ASP Instantiation [177] to improve the efficiency. . The IRIS reasoner [174] re-orders the literals in a rule body to let the most restrictive literals appear first. The preferred order is: positive literals, built-ins, and

negated literals. Reordering techniques has also been applied in query optimization [178-180]. The authors claim "A consequence of the top-down strategy is that a query optimizer for first order queries should place literals that ground many variables as early as possible. Grounding variables decreases the non-determinacy of the literals that follow, which decreases execution time" [180].

What I focus on is backward reasoning for OWL Horst rules over a large knowledge base facing changes. The number of solutions of a goal is not directly available in the knowledge base. The timer to resolve each goal will depend on the size of knowledge base and there are no obvious ways to estimate a-priori the time to resolve goals. Therefore, the reordering techniques discussed above cannot be applied to our backward reasoning directly. I propose a simple way of implementing the selection function to support backward ontology reasoning on a large, changing knowledge base. I propose to select the next goal to be evaluated on the basis of the number of variables. To be specific, we choose to evaluate next the goal with the minimal number of bound arguments.

The body of a rule consists of a conjunction of multiple clauses. Traditional SLD (Selective Linear Definite) clause resolution systems such as Prolog would normally attempt these in left-to-right order, but, logically, we are free to attempt them in any order.

We expect that, given a rule under proof, ordering the body clauses into ascending order by the number of free variables will help to decrease the reasoning time. For example, let us resolve the goal "`?y rdf:type Student`", and consider the rule:

```
[rdfs3: (?x ?p ?y)  (?p rdfs:range ?c) -> (?y rdf:type ?c)]
```

The goal "?y rdf:type Student" matches the head of rule "?y rdf:type ?c", and ?c is unified with Student.

If we select body clause "?x ?p ?y" to prove first, it will yield more than 5 million (using LUBM(40) [161]) instances of clauses. The proof of body clause "?x ?p ?y" in backward chaining would take up to hours. Result bindings of "?p" will be propagated to the next body clause "?p rdfs:range ?c" to yield new clauses (p1 rdfs:range Student), (p2 rdfs:range Student), …, (p32 rdfs:range Student), and then a separate proof would be attempted for each of these specialized forms.

If we select body clause "?p rdfs:range Student" (?c is unified with Student) to prove first, it will yield zero (using LUBM(40)) instances of clauses. The proof of body clause "?p rdfs:range Student" would take up to seconds. No result bindings would be propagated to body clause "?x ?p ?y". The process of proof terminates.

The body clause "?p rdfs:range ?c" has one free variable ?p while the body clause "?x ?p ?y" has three free variables. It is reasonable to prove body clause with fewer free variables first, and then propagate the result bindings to ?p to next body clause "?x ?p ?y". Mostly, goals with fewer free variables cost less time to be resolved than goals with more free variables, since fewer free variables means more bindings and body clauses with fewer free variables will match fewer triples.

5.3.2 Switching between Binding Propagation and Free Variable Resolution

Binding propagation and free variable resolution are two modes of for dealing with conjunctions of multiple goals. I claim that dynamic selection of these two modes

during the reasoning process will increase the efficiency in terms of response time and scalability.

These modes differ in how they handle shared variables in successive clauses encountered while attempting to prove the body of a rule. Suppose that we have a rule body containing clauses `(?x p1 ?y)` and `(?y p2 ?z)` (other patterns of common variables are, of course, also possible) and that we have already proven that the first clause can be satisfied using value pairs $\{(x_1, y_1), (x_2,y_2),\ldots(x_n,y_n)\}$.

In the binding propagation mode, the bindings from the earlier solutions are substituted into the upcoming clause to yield multiple instances of that clause as goals for subsequent proof. In the example given above, the value pairs from the proof of the first clause would be applied to the second clause to yield new clauses `(y1 p2 ?z)`, `(y2 p2 ?z)`, ..., `(yn p2 ?z)`, and then a separate proof would be attempted for each of these specialized forms. Any `(y,z)` pairs obtained from these proofs would then be joined to the `(x,y)` pairs from the first clause.

In the free variable resolution mode, a single proof is attempted of the upcoming clause in its original form, with no restriction upon the free variables in that clause. In the example above, a single proof would be attempted of `(?y p2 ?z)`, yielding a set of pairs $\{(y_n, z_1), (y_{n+1},z_2),\ldots(y_{n+k},z_k)\}$. The join of this with the set $\{(x_1, y_1), (x_2,y_2),\ldots(x_n,y_n)\}$ would then be computed to describe the common solution of both body clauses.

The binding propagation mode is used for most backward chaining systems [98]. There is a direct tradeoff of multiple proofs of narrower goals in binding propagation against a single proof of a more general goal in free variable resolution. As the number of

tuples that solve the first body clause grows, the number of new specialized forms of the subsequent clauses will grow, leading to higher time and space cost overall. If the number of tuples from the earlier clauses is large enough, free variable resolution mode will be more efficient. (In the experimental results in Section 5.5 and Section 5.6, I will demonstrate that neither mode is uniformly faster across all problems.)

Following is an example (using LUBM(40)) showing one common way of handling shared variables between body clauses.

Suppose we have an earlier body clause 1: "?y type Course" and a subsequent body clause 2: "?x takesCourse ?y". These two clauses have the common variable ?y. In our experiments, it took 1.749 seconds to prove body clause 1 while it took an average of 0.235 seconds to prove body clause 2 for a given value of ?y from the proof of body clause 1. However, there were 86,361 students satisfying variable ?x, which means it would take 0.235 *86,361=20,295 seconds to finish proof of 86,361 new clauses after applying value pairs from the proof of body clause 1. 20,295 seconds is not acceptable as query response time. We need to address this problem to improve reasoning efficiency in terms of response time and scalability.

I propose to dynamically switch between modes based upon the size of the partial solutions obtained so far. Let n denote the number of solutions that satisfy an already proven clause. Let t denote the threshold used to dynamically select between modes. If $n \leqslant t$, then the binding propagation mode will be selected. If $n > t$, then the free variable resolution mode will be selected. The larger the threshold is, the more likely binding propagation mode will be selected.

Suppose that we have a rule body containing clauses (a1 p1 b1) (a2 p2

b2). Let (a1 p1 b1) be the first clause, and (a2 b2 c2) be the second clause. ai, bi and ci (i $\in$ [1,2]) could be free variable or concrete value. Assume that there is at least one common variable between two clauses.

In the binding propagation mode, the value pairs from the proof of the first clause would be applied to the second clause to yield new clauses ($a2_1$ $p2_1$ $b2_1$), ($a2_2$ $p2_2$ $b2_2$), ..., ($a2_n$ $p2_n$ $c2_n$), and then a separate proof would be attempted for each of these specialized forms. Any value sets obtained from these proofs would then be joined to the value sets from the first clause. Let $join_1$ denote the time spent on the join operations. Let $proof_1^i$ denote the time of proving first clause with i free variables and $proof_2^j$ be the average time of proving new specialized form with j free variables. (i $\in$ [1,3], j $\in$ [0,2])

In the free variable resolution mode, a single proof is attempted of the upcoming clause in its original form, with no restriction upon the free variables in that clause. A single proof would be attempted of (a2 p2 b2), yielding a set of value sets. The join of the value sets yielded from the first clause and the values sets yielded from the second clause would then be computed to describe the common solution of both body clauses. Let $join_2$ denote the time spent on the join operations. Let $proof_3^k$ denote the time of proving second clause with k free variables (k $\in$ [1,3]).

Determining t is critical to switching between two modes. Let us compare the time spent on binding propagation mode and free variable resolution mode to determine t. Binding propagation is favored when

$$proof_1^i + proof_2^j * n + join_1 < proof_1^i + proof_3^k + join_2$$

Isolating the term involving n,

$proof_2^j * n < proof_1^i + proof_3^k + join_2 - proof_1^i - join_1$

$proof_2^j * n < proof_3^k + join_2 - join_1$

$join_1$ is less than or equal to $join_2$, because the value sets from the second clause in the binding propagation mode have already been filtered by the value sets from the first clause first. The join operations in binding propagation mode are therefore a subset of the join operations in free variable resolution mode. Let t be the largest integer value such that

$proof_2^j * t < proof_3^k$.

Then,

$proof_2^j * t <= proof_2^j * n < proof_3^k + join_2 - join_1$.

We conclude that:

$t = floor(proof_3^k / proof_2^j)$ (1)

Formula (1) provides thus a method for calculating the threshold t that determines when to employ binding propagation. In that formula, k denotes the number of free variables in the second clause (a2 p2 b2), j denotes the number of free variables of the new specialized forms ($a2_1$ $p2_1$ $b2_1$), ($a2_2$ $p2_2$ $b2_2$), (al_n $p2_n$ $c2_n$) of the second clause with ($k \in [1,3]$, $j \in [0,2]$).The specialized form of the second clause has one or two less free variables than the original form. Hence the possible combinations of (k,j) are {(3,2),(3,1),(2,1),(2,0),(1,0)}.

To estimate $proof_3^k$ and $proof_2^j$, we record the time spent on proving goals with different numbers of free variables. We separately keep a record of the number of goals that have one free variable, two free variables and three free variables after we start calling the optimized backwardChaining algorithm. We also record the time spent on

proving these goals. After we have recorded a sufficient number of proof times (experiments will give us an insight into what constitutes a 'sufficient' number), we compute the average time spent on goals with k free variables and j free variables respectively to obtain an estimate of $proof_3^k$ and $proof_2^j$.

In order to adopt accurate threshold to help improve the efficiency, we apply different thresholds to different situations with corresponding number of free variable set (k,j).

We assign the initial value to t from previous experiments in a particular knowledge base/query environment if they exist or zero otherwise.

We update the threshold several times when answering a particular query. The threshold will change as different queries are being answered. For each query, we will call the optimized backward chaining algorithm recursively several times. Each call of backwardChaining is given a specific goal as an input. During the running of backwardChaining, the average time of proving a goal as a function of the number of free variables will be updated after a goal has been proven. During the running of backwardChaining, every time before making selection between two modes the estimate threshold is updated before making the decision.

5.3.3 How to Avoid Repetition and Non-Termination

Given RDFS Rules [115], Horst rules [116] and custom rules [5] in the rule set and queries for answering, backward chaining for ontology reasoning may hit the same goals for several times. Some body clauses such as ?a rdfs:subClassOf ?b and ?x rdfs:subPropertyOf ?y appear in multiple rules in the Horst rule set that is used in many reasoning systems. During the process of answering a given query, these rules

containing the same body clauses might be necessary to be proved to answer the query. During the process of answering a given query, some rules may be repeatedly called for more than one time, leading to proving the same body clause like `?a rdfs:subClassOf ?b` more than one time. Within the process of answering one query, such a repetition decreases the efficiency in terms of response time. Backward chaining with memorization will help to avoid repetition.

Backward chaining is implemented in Logic Programming [181] by SLD resolution [173]. When we apply conventional backward chaining process to ontology reasoning, it has the same non-termination problem as SLD resolution does. During the proving process, the rule body needs to be satisfied to prove the goal. In some cases, the rule body requires proving goals that have the same property as the goal, resulting possibly in an infinite loop unless steps are taken to ensure termination.

For example:

```
[rdfs8:  (?a rdfs:subClassOf ?b), (?b rdfs:subClassOf ?c) -> (?a
rdfs:subClassOf ?c)]
```

is one rule in the RDFS rule set used for ontology reasoning. When we apply standard backward chaining to ontology reasoning, proving the head `(?a rdfs:subClassOf ?c)` requires proving of the body `(?a rdfs:subClassOf ?b)` and `(?b rdfs:subClassOf ?c)`. This loop will be infinite without applying any techniques.

I use an adaptation of the OLDT algorithm to solve this non-termination problem. The OLDT algorithm is an extension of the SLD-resolution [173] with a left to right computation rule. OLDT maintains a solution table and lookup table to solve the non-termination problem.

5.3.4 owl:sameAs Optimization

The "owl:sameAs" relation poses a problem [169] for almost all the reasoning systems including forward chaining. In the reasoning system, we first pre-compute all possible owl:sameAs pairs and save them to a sameAs table. Second, we select a representative node to represent an equivalence class of owl:sameAs URIs. Third, we replace the equivalence class of owl:sameAs URIs with the representative node. At last, if users want to return all the identical results, we populate the query response using the sameAs table by replacing the representative node with the URIs in the equivalence class.

As I described in Section 5.1, reasoning with the owl:sameAs relation can result in a multiplication of the number of instances of variables during backward-chaining and expanded patterns in the result. As long as that triple is in the result set, all of the members in its equivalence class would be in the result set as well. This adds cost to the reasoning process in both time and space. The optimization that applies pre-computation and selects a representative node improves the performance in terms of time and space.

This optimization is a novel adaptation of owl:sameAs optimization from forward chaining reasoning systems, such as OWLIM-SE [90] and Oracle [91], to backward chaining reasoning systems.

5.4 Evaluation of Optimized Backward Chaining

In the previous section, I discussed four optimizations that have been introduced in the optimized backward chaining algorithm: optimized selection function, dynamic switching between binding propagation mode and free variable resolution mode,

avoidance of repetition and non-termination (application of OLDT) and "owl:sameAs" optimization. In this section, I present experimental evidence of the effectiveness of the first two optimizations. I do not explore the effectiveness of OLDT as this is now well-established [170] although I apply the same idea of this optimization to the backward chaining algorithm to avoid the explosion of the size of the search space. The benchmark I am using for the experiments in this thesis exclude all the "owl:sameAs" semantics. The owl:sameAs optimization technique would work well with UOBM which includes the "owl:sameAs" semantics; experimentation on the use the technique is left for future work.

All the experiments in this section were performed on a PC with a 2.80 GHz Intel Core i7 processor and 8 G memory, running Windows 7 Enterprise. Sun Java 1.6.0 was used for Java-based tools. The maximum heap size was set to 512M. I checked all of our results for being complete and sound. All the timing results I present in this thesis except Section 5.5 are CPU times as the knowledge base is entirely in memory.

To evaluate the performance of the clause selection function and the dynamic propagation selection, I evaluate the optimized backward chaining by turning these optimizations on and off individually and comparing runs with a technique turned on against runs with the technique turned off. In this section, I only present the scalability and response time of the optimized backward chaining algorithm when it runs in memory without any support of external storage. I am aware that working in memory has limitations with respect to the size of the knowledge base and the retrieved data. I will explore the efficiency of the optimized backward chaining algorithm with external triple storage in Section 5.5.

5.4.1 Selection Function

Table XVIII compares the backward chaining algorithm with the clause selection based on free variable count to the traditional left-to-right selection on a relatively small knowledge base (100,839 triples), LUBM(1) [161]. Backward chaining with the ordered selection function yields considerably smaller query response times for all the queries than left-to-right.

TABLE XVIII
EVALUATION OF CLAUSE SELECTION OPTIMIZATION ON LUBM(1)

	Time (ms), Ordered	Time (ms), Left-to right	Result Size (triples)
Query1	93	605,907	4
Query2	280	2,316,178	0
Query3	0	417,396	6
Query4	452	2,137,151	34
Query5	80	262,924	719
Query6	374	434,665	7,790
Query7	187	1,083,114	67
Query8	514	2,032,895	7,790
Query9	171	1,322,701	208
Query10	78	676,498	4
Query11	213	571,540	224
Query12	250	1,582,130	15
Query13	24	424,931	1
Query14	15	404,884	5,916

The difference becomes even more dramatic for a larger knowledge base (1,272,871 triples), LUBM(10), as shown in Table XIX. With left-to-right selection, we are unable to answer any query within 30 minutes, and out-of-memory errors occur for almost half of the queries. Were the knowledge base moved to external triple storage, the I/O time of accessing the external triple storage would magnify the problem of left-to-right selection.

TABLE XIX
EVALUATION OF CLAUSE SELECTION OPTIMIZATION ON LUBM(10)

	Time (ms), Ordered	Time (ms), Left-to right	Result Size (triples)
Query1	343	OutOfMemoryError: Java heap space	4
Query2	1,060	>1.8*106	28
Query3	15	>1.8*106	6
Query4	858	>1.8*106	34
Query5	15	>1.8*106	719
Query6	1,170	OutOfMemoryError	99,566
Query7	1,341	OutOfMemoryError	67
Query8	1,684	OutOfMemoryError	7,790
Query9	1,591	OutOfMemoryError	2,540
Query10	982	OutOfMemoryError	4
Query11	93	>1.8*106	224
Query12	109	>1.8*106	15
Query13	0	>1.8*106	33
Query14	156	>1.8*106	75,547

5.4.2 Dynamic Selection of Propagation Mode

I compare the backward chaining algorithm with three different modes of resolving goals on LUBM(10) in Table XX. The first mode uses dynamic selection between binding propagation mode and free variable resolution mode. The second mode uses binding propagation mode only. The third mode uses free variable resolution mode only.

Table XX shows that neither binding propagation mode nor free variable resolution mode is uniformly better than the other on all cases. From query1 to query5 and query11 to query14, dynamic mode performs almost same as binding propagation mode. From query5 to query9, dynamic mode performs dramatically better than binding propagation mode with much less query response time. For query1, query3 and query14 only, dynamic mode performs almost same as free variable resolution mode. For the other queries, dynamic mode performs dramatically better than free variable resolution

mode with much less query response time.

TABLE XX
EVALUATION OF DYNAMIC SELECTION VERSUS BINDING PROPAGATION AND FREE VARIABLE MODES ON LUBM(10)

	Time (ms), Dynamic selection	Time (ms), Binding propagation only	Time (ms), Free variable resolution only
Query1	343	343	296
Query2	1,060	1,341	21,278
Query3	15	20	15
Query4	858	961	42,572
Query5	15	16	22,323
Query6	1,170	592,944	19,968
Query7	1,341	551,822	20,217
Query8	1,684	513,773	40,061
Query9	1,591	524,787	20,841
Query10	982	509,078	19,734
Query11	93	109	19,141
Query12	109	156	38,313
Query13	0	10	21,528
Query14	156	140	140

The query response times of query6 to query10 are less by orders of magnitude when running our algorithm with the dynamic selection mode in comparison compared to running with binding propagation mode only and free variable resolution mode only. In all cases the optimized version finishes faster than the better of the other two versions. Overall, the results in Table XX confirm the advantage of dynamically selecting between propagation modes.

5.4.3 Overall Performance

In the semantic web, frequent changes in the underlying knowledge base happen because of continuous harvesting of new facts such as papers published. Less frequent will be changes in the underlying ontology or the rule set that governs the reasoning.

Individuals might be changing rules rather frequently when they are trying to develop the best rules to describe a new concept they want to measure, or instance, 'ground-breaking'. Queries may be posted during the frequent changes. A backward chaining reasoner can handle a change to the knowledgebase without recourse to materialization as a forward chaining reasoner will have to do unless it is willing to provide potentially incomplete or simply wrong answers.

We are exploring the scenario of responding to a query after a change to the knowledge base has occurred and that a forward chaining reasoner will do a new materialization to accommodate the change. I understand that different knowledge bases will encounter this scenario at different frequencies, depending on the rate at which changes occur. I also understand that forward chaining reasoners might be willing to wait and batch individual changes at the prize of incomplete answers for some queries (which may not happen during the period of accumulating the changes), However, for simplicity sake, we will perform the experiments under the assumption of immediate materialization and leave the modeling the dynamic nature of queries and changes over time for future work.

I compare the optimized backward chaining reasoner with OWLIM-SE [90], which is considered to be among the best performers on both scalability and response time. OWLIM-SE is a semantic reasoner that adopts the materialization (forward chaining) mechanism [90]. I use the LUBM Benchmark dataset scaling from one university to 40 universities. The size of the benchmark dataset scales from 100,839 to 5,307,754. There are altogether 14 queries in the benchmark. A semantic reasoner that adopts the materialization (forward chaining) mechanism would load datasets every time

when a change occurs in the system.

TABLE XXI
OVERALL COMPARISON BETWEEN THE BACKWARD CHAINING REASONER AND OWLIM-SE

	LUBM(1)		LUBM(40)	
	Time (ms), Opt. Backwd	Time (ms), OWLIM-SE	Time (ms), Opt. Backwd	Time (ms), OWLIM-SE
Loading time	2,900	9,600	95,000	350,000
Query1	260	27	1,400	26
Query2	490	3.4	9,100	5,100
Query3	56	1.0	36	2.5
Query4	470	8.4	5,900	14
Query5	33	59	15	41
Query6	180	240	43,000	5,300
Query7	190	4.4	51,000	54
Query8	540	460	57,000	3,000
Query9	250	63	87,000	4,400
Query10	140	0.10	51,000	0.60
Query11	190	4.9	200	5.4
Query12	220	1.0	3,600	11
Query13	28	0.20	33	17
Query14	24	23	1,200	2,500

In the Table XXI, for LUBM(1), we can see that the optimized backward chaining reasoner is close to the response time of OWLIM-SE, however, our loading time (initializing the knowledge base - in memory) is only about 1/3 of that of OWLIM-SE's loading time (initialization and materialization). Thus, the backward chaining reasoner performs better than OWLIM-SE on LUBM(1) in cases where we anticipate frequent changes. For LUBM(40), from query6 to query10, the response time of the optimized backward chaining reasoner is far more than OWLIM-SE, however, if we take loading time for OWLIM-SE into consideration, the optimized reasoner still has a better performance.

5.5 Evaluation with External Storage

In Section 5.3 and 5.4, assessing the effectiveness of the optimized reasoner [182], all our experiments were performed 'in-memory' which limited the study to a knowledge base of less than 10 Million triples.

In this section, I switch to implementations that use external storage for the knowledge base. I consider Jena SDB [183], Jena TDB [142] and OWLIM-SE [90]. I extend our study based on a knowledge base of more than 10 Million triples.

The employment of external storage introduces new factors and has implications on how to improve the scalability of the backward chaining reasoner. First, any optimization technique needs to balance the number of accesses to data and the size of the retrieved data against the size of in-memory cache and its use. Second, the algorithm has to take now into account that it will take longer to access a triple (or a set of triples) due to having to perform I/O. In-memory reasoners typically have a 'model' of the knowledge base in which they store the facts and an API to access them. When an external storage is used they would provide transparent connections from the model to the external databases that would allow the reasoner to use the same API for accessing the model. This leads to a third factor effecting the scalability and performance of the reasoner: the middleware that realizes the transparent linking.

Jena SDB provides persistent triple stores using relational databases. An SQL database is required for the storage and query of triples for SDB. In this section, I use MySQL and PostgreSQL as the relational database for SDB. Jena TDB is claimed as a more scalable and faster triple store than SDB [183].A special Jena adapter permits access to OWLIM-SE repositories [90]. Reasoners can access all three storage systems

via a common Jena API.

5.5.1 Preliminary Analysis

I begin by exploring the relative impact on overall performance of the three major components of the backward chaining reasoner, the middleware, and the storage system itself. The purpose of this analysis is to determine how much time we can save by improving any one of these subsystems in isolation.

I employed Jena SDB + MySQL as the external storage for the backward chaining reasoner in the experiment, evaluating the query response time of 14 queries from LUMB [161] using LUBM(30).

A single function in the backward chaining algorithm implementation is responsible for all data retrievals from the triple store. We refer to this function as "the Data-retrieval function" in the remainder of this section. We recorded the clock time T_f and CPU time t_f spent within the Data-retrieval function and in the whole query processing (T_{tot} and t_{tot}, respectively) in Table XXII.

The portion of the CPU and clock times spent in answering the query but not spent in the Data-retrieval function is attributable to the backward chaining reasoner:

$$T_{bw} = T_{tot} - T_f$$

$$T_{bw} = t_{tot} - t_f$$

The clock time observed during the Data-retrieval function includes actual input operations on the underlying triple store, together with the CPU-intensive manipulation of the input data by the middleware layer. Assuming that the ratio, $\rho = t_{tot}/T_{tot}$, of CPU time to clock time observed over the processing of an entire query would remain approximately constant during the middleware CPU, we were able to estimate the portion

of the Data-retrieval function clock time that was attributable to the middleware:

$$T_{mid} = \rho \; t_{mid}$$

and can attribute the remaining clock time as the actual time spent doing I/O:

$$T_{IO} = T_f - T_{mid}$$

Then we can estimate a minimal clock time to answer the query, assuming 100% CPU utilization, as

$$T_{min} = t_{bw} + \rho \; T_{mid} + T_{IO}$$

TABLE XXII
CLOCK TIME, CPU TIME AND I/O TIME FROM EXPERIMENTS WITH JENA SDB USING LUBM(30)

	Total Clock time, T_{tot} (ms)	Total CPU Time, t_{tot} (ms)	Clock time in Data-retrieval function, T_f(ms)	CPU time in I/O function, t_f(ms)
Query1	1405.00	951.00	920.00	546.00
Query2	9631.00	6084.00	5058.00	2293.00
Query3	203.00	78.00	109.00	31.00
Query4	35354.00	8096.00	31140.00	5070.00
Query5	173.00	78.00	94.00	15.00
Query6	23744.00	7035.00	19984.00	3712.00
Query7	24058.00	9984.00	18659.00	6333.00
Query8	28694.00	11029.00	22680.00	5896.00
Query9	29598.00	11700.00	23899.00	6988.00
Query10	18612.00	6630.00	15040.00	3572.00
Query11	3636.00	561.00	2964.00	124.00
Query12	7567.00	1903.00	5226.00	405.00
Query13	187.00	46.00	95.00	0.00
Query14	1873.00	811.00	1451.00	452.00

Table XXIII shows the values of these estimates, together with the percentage of that value attributable to each of the three components. In Table XXIII, the percentage of time spent in I/O operations ranges from 22% to 75%, a considerable variation. This might be because some retrievals from triple store retrieve huge numbers of triples while others are far more focused and process much less data.

TABLE XXIII
ESTIMATED I/O TIME AND IDEAL PERCENTAGES FROM EXPERIMENTS WITH JENA SDB USING LUBM(30)

	Min possible clock time to answer a query, T_{min} (ms)	**% of T_{min} spent in I/O**	**% of Tmin spent in BW chaining**	**% of T_{min} time spent in middleware**
Query1	1217.15	0.22	0.33	0.45
Query2	8376.00	0.27	0.45	0.27
Query3	125.00	0.38	0.38	0.25
Query4	32175.53	0.75	0.09	0.16
Query5	153.19	0.49	0.41	0.10
Query6	22818.84	0.69	0.15	0.16
Query7	19277.93	0.48	0.19	0.33
Query8	26801.04	0.59	0.19	0.22
Query9	27147.26	0.57	0.17	0.26
Query10	17497.60	0.62	0.17	0.20
Query11	3334.32	0.83	0.13	0.04
Query12	6496.09	0.71	0.23	0.06
Query13	141.00	0.67	0.33	0.00
Query14	1730.68	0.53	0.21	0.26

The percentage of the time devoted to the middleware ranges from 0% to 44%, with an average around 20%, indicating that the triple storage layer adds a significant component of CPU time. The backward chaining code running on top of that accounts for 13 to 45% of minimal processing time, and the average is 25%.

These percentages are surprisingly balanced, suggesting that improvements to any one of the three major components of the system can have only modest effect on the total time. Significant improvements will be possible only by improvement in all three areas. One possible avenue of exploration is changes to the reasoner that would not only speed up the reasoner but would affect the number and size of requests for input from the underlying store. Indirectly, at least, several of the optimizations I have proposed in Section 5.3 could have such an effect. Caching, an effect not explored in this experiment, could also have a major impact across all three areas.

5.5.2 Evaluation of the Optimization Techniques

In this section, I examine the impact of the two major optimizations proposed in Section 5.3.

Ordered Selection Function

I replace the traditional left-to-right processing of clauses within rule bodies by ordering by ascending number of free variables.

Table XXIV compares the backward chaining algorithm with the clause selection based on free variable count to the traditional left-to-right selection on a relatively small knowledge base (100,839 triples), LUBM(1) [161], stored in Jena TDB. Backward chaining with the ordered selection function yields considerably smaller query response times for all the queries than left-to-right. The I/O time of accessing the external triple storage magnifies the problem of left-to-right selection compared to [182] because the knowledge base is in external triple storage TDB now.

TABLE XXIV
EVALUATION OF CLAUSE SELECTION OPTIMIZATION ON LUBM(1) USING TDB AS EXTERNAL STORAGE

	Time (ms), Ordered	Time (ms), Left-to right	Result Size (triples)
Query1	296	>6.0*105	4
Query2	811	>6.0*105	0
Query3	46	>6.0*105	6
Query4	1419	>6.0*105	34
Query5	31	>6.0*105	719
Query6	265	>6.0*105	7,790
Query7	234	>6.0*105	67
Query8	483	>6.0*105	7,790
Query9	202	>6.0*105	208
Query10	156	>6.0*105	4
Query11	218	>6.0*105	224
Query12	202	>6.0*105	15
Query13	15	>6.0*105	1
Query14	31	>6.0*105	5,916

The difference becomes even more dramatic for a larger knowledge base (1,272,871 triples), LUBM(10) stored in Jena TDB, as shown in Table XXV. With left-to-right selection, we are unable to answer any query within 30 minutes, and out-of-memory errors occur for almost half of the queries. The I/O time of accessing the external triple storage magnifies the problem of left-to-right selection compared to [182] because the knowledge base is in external triple storage TDB now.

Switching between Binding Propagation and Free Variable Resolution

Binding propagation and free variable resolution are two modes for dealing with conjunctions of multiple goals. I have proposed dynamic selection of these two modes during the reasoning process to increase the efficiency in terms of response time and scalability.

I compare the backward chaining algorithm with three different modes of resolving goals on LUBM(10) stored in Jena TDB in Table XXVI. The first mode uses

dynamic selection between binding propagation mode and free variable resolution mode. The second mode uses binding propagation mode only. The third mode uses free variable resolution mode only.

TABLE XXV
EVALUATION OF CLAUSE SELECTION OPTIMIZATION ON LUBM(10) USING TDB AS EXTERNAL STORAGE

	Time (ms), Ordered	Time (ms), Left-to right	Result Size (triples)
Query1	1045	OutOfMemoryError: Java heap space	4
Query2	2433	>2.0*106	28
Query3	31	>2.0*106	6
Query4	3744	>2.0*106	34
Query5	15	>2.0*106	719
Query6	1435	OutOfMemoryError	99,566
Query7	1903	OutOfMemoryError	67
Query8	2106	OutOfMemoryError	7,790
Query9	1918	OutOfMemoryError	2,540
Query10	1138	OutOfMemoryError	4
Query11	140	>2.0*106	224
Query12	358	>2.0*106	15
Query13	15	>2.0*106	33
Query14	187	>2.0*106	75,547

Table XXVI shows that neither binding propagation mode nor free variable resolution mode is uniformly better than the other on all cases. From query 1 to query 5 and in query 13, dynamic mode performs almost same as binding propagation mode. From query 6 to query 10, dynamic mode performs dramatically better than binding propagation mode with much less query response time. For query 11, query 12 and query 14, dynamic mode performs better than binding propagation mode with less query response time. For query1, query3 and query 14 only, dynamic mode performs almost same as free variable resolution mode. For the other queries, dynamic mode performs dramatically better than free variable resolution mode with much less query response

time. The query response times of query6 to query10 are less by orders of magnitude when running the algorithm with the dynamic selection mode in comparison compared to running with binding propagation mode only and free variable resolution mode only. In all cases the optimized version finishes faster than the better of the other two versions.

TABLE XXVI
EVALUATION OF DYNAMIC SELECTION VERSUS BINDING PROPAGATION AND FREE VARIABLE MODES ON LUBM(10) USING TDB AS EXTERNAL STORAGE

	Time (ms), Dynamic selection	Time (ms), Binding propagation only	Time (ms), Free variable resolution only
Query1	1045	904	904
Query2	2433	2683	26535
Query3	31	15	15
Query4	3744	4149	41605
Query5	15	15	2244810
Query6	1435	>6.0*105	20514
Query7	1903	>6.0*105	20763
Query8	2106	>6.0*105	42831
Query9	1918	>6.0*105	21512
Query10	1138	>6.0*105	19921
Query11	140	904	19094
Query12	358	1435	41745
Query13	15	31	24117
Query14	187	1154	187

Overall, the results in Table XXVI confirm the advantage of dynamically selecting between propagation modes. The I/O time of accessing the external triple storage magnifies the problem of binding propagation mode only and free variable resolution mode only compared to [182] because the knowledge base are in external triple storage TDB now. The selection of the threshold in dynamic mode is affected by the employment of external storage and affects the number of accesses to store.

Storage System Impact

To explore the effect of switching the underlying storage manager, I compared three external storage methods employed in the optimized backward chaining reasoner with regard to I/O time. For all 14 queries from LUBM, the three storage managers SDB, TDB and OWLIM-SE, all have the same number of accesses (calls to the Data-retrieval function) to the underlying store.

I show in Table XXVII the I/O time per access for SDB, TDB and OWLIM-SE using LUBM(50) which has 6,890,640 triples. The I/O time per store access of SDB is dramatically longer than both TDB and OWLIM-SE through all 14 queries in LUBM. From query 1 to 5 and query 13, the I/O time per store access of TDB is slightly longer than OWLIM-SE. For the other queries, TDB has shorter I/O time per store access. In general, TDB and OWLIM-SE have similar performance in terms of I/O time.

TABLE XXVII
COMPARISON AMONG SDB, TDB AND OWLIM-SE AS EXTERNAL STORAGE ON I/O TIME PER STORE ACCESS ON LUBM(50)

	Time (ms), SDB+ PostgreSQL	Time (ms), TDB	Time (ms), OWLIM-SE	#of Number of access to store
Query1	41.42	2.32	0.70	132
Query2	50.76	0.48	0.35	353
Query3	1.63	0.42	0.28	65
Query4	82.38	0.38	0.14	455
Query5	1.57	0.36	0.20	81
Query6	298.74	0.67	5.12	153
Query7	237.69	0.13	0.52	286
Query8	72.24	0.07	0.43	917
Query9	221.45	0.02	0.17	351
Query10	223.33	0.07	0.14	218
Query11	2.08	0.05	0.12	616
Query12	2.07	0.03	0.10	2792
Query13	1.28	0.21	0.13	86
Query14	111.76	0.03	0.22	67

Overall Performance

Finally, I consider the overall performance of the optimized backward chaining reasoner when using three different storage managers. I use SDB, TDB and OWLIM-SE respectively when running the optimized backward chaining reasoner for all 14 queries from LUBM, and I measure query response time using LUBM(50) in Table XXVIII. Query1, query2, query3 and query6, OWLIM-SE has the fastest response time. Jena SDB + PostgreSQL performs fastest only for query4, because the I/O time of Jena SDB is the longest out of three stores. For the rest of the queries, Jena TDB is fastest.

TABLE XXVIII
COMPARISON BETWEEN SDB, TDB AND OWLIM-SE AS EXTERNAL STORAGE ON QUERY RESPONSE TIME ON LUBM(50)

	Clock Time		
	Time (ms), SDB+PostgreSQL	**Time (ms), TDB**	**Time (ms), OWLIM-SE**
Query1	6430	13440	3549
Query2	24960	36102	17046
Query3	406	58	61
Query4	46400	71298	45680
Query5	533	78	156
Query6	59144	32590	30470
Query7	83799	34580	45527
Query8	85563	48307	53013
Query9	95992	34583	49566
Query10	63100	20191	27916
Query11	3466	528	876
Query12	16253	2403	3199
Query13	374	39	37
Query14	8581	4731	5364

In Table XXIX I show a similar comparison of TDB and OWLIM-SE on query response time using LUBM(100). SDB was omitted from this comparison because the loading time of SDB is prohibitively long. In both Table XXVIII and Table XXIX, Jena

TDB has the better performance through all 14 queries. In general, the optimized backward chaining reasoner and external storage Jena TDB has the best performance especially when the size of the knowledge base increases.

TABLE XXIX
COMPARISON BETWEEN TDB AND OWLIM-SE AS EXTERNAL STORAGE ON QUERY RESPONSE TIME ON LUBM(100)

	Clock Time	
	Time (ms), TDB	Time (ms), OWLIM-SE
Query1	2652	5085
Query2	13884	29657
Query3	31	46
Query4	49109	82664
Query5	46	78
Query6	26020	51277
Query7	39873	76752
Query8	58609	98343
Query9	46925	85456
Query10	26894	52821
Query11	452	826
Query12	920	1716
Query13	15	31
Query14	7222	11263

5.6 Evaluation with Custom Rule Sets and Queries

5.6.1 Ontology Data, Custom Rule Sets and Queries

TABLE XXX
Size Range of Datasets (in Triples)

	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**	**Dataset5**	**Dataset6**
ScienceWeb	3511	6728	13244	166163	332248	1327573
LUBM	8814	15438	34845	100838	624827	1272870
	Dataset7	**Dataset8**	**Dataset9**	**DataSet10**	**DataSet11**	
ScienceWeb	2656491	3653071	3983538			
LUBM	2522900	4109311	6890949	13880279	27643953	

The size range of the datasets in our experiments is listed in Table XXX. I generate 11 datasets for LUBM and 9 datasets for ScienceWeb, both ranges from thousand to millions for our experiments. I have added 4 more datasets from LUBM to the ontology data in Chapter 3 for the experiments in this section.

Below I will give the 5 rule sets and 3 corresponding query sets that I will use in the experiments. I have made some changes to the rule sets and queries that I have introduced in section3. Rule sets were defined to test basic reasoning to allow for validation, such as allowing for transitivity and recursion. Rule set 1 for the co-authorship relation, rule set two is for collaborator relation, rule set three is used in queries for the genealogy of PhD advisors (transitive) and rule set 4 is to enable queries for "good" advisors. Rule set 5 is a combination of the first 4 sets.

Rule set 1: Co-author

```
authorOf(?x, ?p)∧authorOf(?y, ?p) ⇒ coAuthor(?x, ?y)
```

Rule set 2: Collaborator

```
advisorOf (?x, ?y) ⇒ collaboratorOf (?x, ?y)
```

Rule set 3: Research ancestor (transitive)

```
advisorOf(?x, ?y) ⇒ researchAncestor(?x, ?y)
researchAncestor(?x, ?y)∧researchAncestor(?y, ?z) ⇒
researchAncestor(?x, ?z)
```

Rule set 4: Distinguished advisor (recursive)

```
advisorOf(?x,?y)∧advisorOf(?x,?z)∧notEqual(?y,?z)
∧worksFor(?x,?u) ⇒ distinguishAdvisor(?x, ?u)
advisorOf(?x,?y)∧distinguishAdvisor(?y,?u)∧worksFor(?x,?d)
⇒ distinguishAdvisor(?x, ?d)
```

Rule set 5: combination of above 4 rule sets.

I have composed 3 query sets to use in the tests, expressed in SPARQL notation:

Query set1:

Query 1: Co-author

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
```

```
WHERE {?x uni:coAuthor ?y.};
```

Query 2: Collaborator

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:collaboratorOf?y. };
```

Query 3: Research ancestor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:researchAncestor ?y.};
```

Query 4: Distinguished advisor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:distinguishAdvisor ?y. };
```

Query set2:

Query 1: Co-author

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:coAuthor ?y. ?x uni:hasName
\"FullProfessor0_d0_u0\" }
```

Query 2: Collaborator

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni: collaboratorOf ?y. ?x uni:hasName
\"FullProfessor0_d0_u0\" };
```

Query 3: Research ancestor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:researchAncestor ?y. ?x uni:hasName
\"FullProfessor0_d0_u0\" };
```

Query 4: Distinguished advisor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:distinguishAdvisor ?y. ?y uni:hasTitle
\"department0u0\" };
```

Query set3:

Query 1: Co-author

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {<http://www.d0.u0.edu/~FullProfessor0_d0_u0> uni:coAuthor
?y. }
```

Query 2: Collaborator

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {<http://www.d0.u0.edu/~FullProfessor0_d0_u0>  uni:
collaboratorOf ?y. };
```

Query 3: Research ancestor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {<http://www.d0.u0.edu/~FullProfessor0_d0_u0>
uni:researchAncestor ?y. };
```

Query 4: Distinguished advisor

```
PREFIX uni:<http://www.owl-
ontologies.com/OntologyUniversityResearchModel.owl#>
SELECT ?x  ?y
WHERE {?x uni:distinguishAdvisor <http://www.d0.u0.edu>.};
```

There are minor differences among the above three query sets. Query set 1 is intended to retrieve all the pairs of relationships, for example, all the co-authors in the knowledge base. Query set 2 is intended to retrieve partial pairs of relationships, for example, all the co-authors of researchers whose name is "FullProfessor0_d0_u0". Query set 3 is intended to retrieve pairs of relationships for a specific researcher/department, for example, all the co-authors of researcher

<http://www.d0.u0.edu/~FullProfessor0_d0_u0>.

Query set 2 is the query form we used in paper [5]. For query set 2, the optimized backward chaining reasoner answers two separate queries and then applies a join operation resulting in a cross product. For query set 1 and 3, the optimized backward

chaining reasoner only need to answer one single query respectively.

Queries are used with the rules sets that define the properties employed in the queries. Each rule set is tested with corresponding queries in different query sets. Rule set 5 is tested with all queries.

5.6.2 Experimental Environment and Metrics

I have chosen Jena TDB as our external storage support for the optimized backward chaining reasoner. The latest version systems have been chosen: Jena (2.11.0, 2013-09-18 release), and Jena TDB (1.0.0, 2013-09-18 release). Consider that a backward chaining system does not require expensive up front closure computation every time the knowledge base changes, I have taken scalability and query processing time from [5] as the main metrics.

- Query processing time: This stage starts with parsing and executing the query and ends when all the results have been saved in the result set. It includes the time of traversing the result set sequentially.

All the experiments in this section were performed on a PC with a 2.80 GHz Intel Core i7 processor and 8 G memory, running Windows 7 Enterprise. Sun Java 1.6.0 was used for Java-based tools. The maximum heap size was set to 512M. I checked all of our results for being complete and sound. All the timing results I present in this section are CPU times as the knowledge base is entirely in memory.

5.6.3 Evaluation Procedure

Our goal is to evaluate the performance of the optimized backward chaining reasoner in terms of reasoning and querying time using custom rules. I am interested in two aspects of performance. One aspect is scalability, which means the size of data and

the complexity of reasoning. The second aspect is query processing time. I am interested in the query processing time as the size of the knowledge base changes from small toy size to realistic sizes of millions.

5.6.4 Results and Discussion

Evaluation on top of LUBM

With rule sets and query sets described in Section 5.6.1, I evaluate the backward chaining reasoner on 11 datasets generated from LUBM. In this section, all the datasets are stored in external storage using Jena TDB as our support. The evaluation results of the backward chaining reasoner on top of LUBM is shown in Table XXXI focus on two aspects of evaluation on supporting reasoning of customized rules. The first aspect is scalability. With support of external storage, the optimized backward chaining reasoner can handle up to about 30 million size dataset in our experimental environment. The second aspect is performance in terms of query processing time. As the size of dataset increases, the query processing time scales from less than 1 second to about half minute.

As shown in Table XXXI, for all the datasets, query processing time of all queries from query set 1 are slightly better than query set 2 because of the minor difference that I have discussed in Section 5.2.6. The optimized backward chaining reasoner applies join operation to answer queries from query set 2. We believe that the other ontology reasoning systems may employ different methods in query processing to answer multi-queries.

For all the datasets, query processing time of all queries from query set 3 is less than 1 seconds. I do not present the evaluation results of query set 3 in this thesis. Compared with results in paper [5] for query set 2, the optimized backward chaining

reasoner has better scalability than Jena, Kaon2 and Pellet. From dataset 1 to dataset 6, we can see that the optimized backward chaining reasoner has almost the same query processing time as OWLIM and Oracle. The backward chaining reasoner performs reasoning at query time. Thus, as the size of dataset increases, Oracle and OWLIM requires less time in query processing than the backward chaining reasoner. However, both Oracle and OWLIM requires expensive up front closure computation when knowledge base changes. Thus, the backward chaining reasoner performs better than OWLIM and Oracle in cases where we anticipate frequent changes.

TABLE XXXI
QUERY PROCESSING TIME (MS) FOR QUERY SET1 AND QUERY SET2 ON LUBM

	Query set1	Query set2	Query set1	Query set2	Query set1	Query set2	Query set1	Query set2
	Query1		Query2		Query3		Query4	
Dataset1	171	249	124	171	109	156	156	249
Dataset2	234	265	159	187	140	187	249	327
Dataset3	421	468	171	249	171	265	374	452
Dataset4	733	889	265	452	421	452	592	764
Dataset5	1669	1778	811	951	858	1029	1591	1606
Dataset6	2823	2979	1232	1435	1388	1497	2293	2386
Dataset7	4570	4836	1934	2168	2308	2324	3447	3478
Dataset8	6115	6832	2808	2839	2917	3244	4602	4655
Dataset9	10233	11044	4258	4586	4664	5116	6848	6879
Dataset 10	19012	19671	7534	7960	8938	9250	11793	12074
Dataset 11	37034	37752	15241	16536	18064	18392	21902	22932

Evaluation Using the ScienceWeb Ontology

With rule sets and query sets described in Section 5.2.6, I evaluate the backward chaining reasoner on 9 datasets generated from ScienceWeb ontology. In this section, all

the datasets are stored in external storage using Jena TDB as our support. The evaluation results of the backward chaining reasoner on top of ScienceWeb ontology is shown in Table XXXII. For scalability, with support of external storage, the optimized backward chaining reasoner can handle all of 9 datasets in our experimental environment. For performance in terms of query processing time, as the size of dataset increases, the query processing time scales from less than 1 second to about 10 seconds.

As shown in Table XXXII, for all the datasets, query processing time of all queries from query set 1 are slightly better than query set 2 because of the minor difference that I have discussed in Section 5.2.6. The optimized backward chaining reasoner applies join operation to answer queries from query set 2.

TABLE XXXII
QUERY PROCESSING TIME FOR QUERY SET1 AND QUERY SET2 ON SCIENCE ONTOLOGY UNIT: MS)

	Query set1	Query set2	Quer y set1	Quer y set2	Query set1	Quer y set2	Query set1	Query set2
	Query1		Query2		Query3		Query4	
Dataset1	124	156	93	109	296	358	93	156
Dataset2	140	187	109	124	436	483	109	171
Dataset3	187	280	124	171	156	249	124	202
Dataset4	780	811	358	499	873	967	530	717
Dataset5	1107	1232	639	670	1404	1450	982	1201
Dataset6	2090	2449	1014	1092	3104	3603	1435	1887
Dataset7	3806	4040	1466	1716	4976	5116	2574	2683
Dataset8	5288	5382	1887	1903	8767	8798	3744	3822
Dataset9	5834	6115	2168	2449	10483	10670	4851	4929

Similar to the discussion in previous section, for all the datasets, query processing time of all queries from query set 3 is less than 1 seconds. I do not present the evaluation results of query set 3 in this thesis.

Compared with results in paper [5] for query set 2, the optimized backward chaining reasoner has better scalability than Jena, Kaon2 and Pellet. From dataset 1 to dataset 5, we can see that the optimized backward chaining reasoner has almost the same query processing time as OWLIM and Oracle. The backward chaining reasoner performs reasoning at query time. Thus, as the size of dataset increases, Oracle and OWLIM requires less time in query processing than the backward chaining reasoner. However, both Oracle and OWLIM requires expensive up front closure computation when knowledge base changes. Thus, the backward chaining reasoner performs better than OWLIM and Oracle in cases where we anticipate frequent changes.

Comparison between In-memory Store and External Storage on LUBM

I employed Jena TDB for the support of external storage in order to increase the scalability of the optimized backward chaining reasoner. We anticipate that some applications might want to balance performance and scalability. For example, for some mobile applications, performance is more important than scalability.

In this section, I compare the performance of the optimized backward chaining reasoner with and without support of external storage on all the queries from query set1 and query set2. The evaluation results are presented in Table XXXIII.

When the optimized backward chaining reasoner runs in memory without any support of external storage, it can only handle up to 7 data sets from LUBM. Thus I only compared performance on 7 data sets. I am aware that working in memory has limitations with respect to the size of the knowledge base and the retrieved data. As Table XXXIII shows, our reasoner running with support of Jena TDB has twice the processing time as much as running entirely in memory, that is, running entirely in memory performs better

than running with support of Jena TDB. For light applications like mobile applications, in-memory version would be a better choice.

TABLE XXXIII
COMPARISON OF QUERY PROCESSING TIME BETWEEN IN-MEMORY STORE AND EXTERNAL STORAGE ON LUBM(UNIT: MS)

Query	Type	Query set	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5	Dataset 6	Dataset 7
Query1	TDB	set1	171	234	421	733	1669	2823	4570
		set2	249	265	468	889	1778	2979	4836
	In memory	set1	93	109	124	358	748	1107	2059
		set2	109	140	171	363	858	1232	2199
Query2	TDB	set1	124	159	171	265	811	1232	1934
		set2	171	187	249	452	951	1435	2168
	In memory	set1	46	46	62	93	249	296	421
		set2	78	93	98	140	296	374	530
Query3	TDB	set1	109	140	171	421	858	1388	2308
		set2	156	187	265	452	1029	1497	2324
	In memory	set1	56	62	78	124	296	390	639
		set2	78	109	113	171	358	483	733
Query4	TDB	set1	156	249	374	592	1591	2293	3447
		set2	249	327	452	764	1606	2386	3478
	In memory	set1	93	140	202	327	1107	1263	1887
		set2	156	202	296	343	1154	1279	1905

CHAPTER 6

TRUST-BASED HYBRID REASONING

In this chapter, I explore the idea of "trust", where each change to the knowledge base is analyzed as to what subset of the knowledge base is impacted by the change and could therefore contribute to incorrect inferences. I will present algorithms that adapt the reasoner such that, when proving a goal, it does a simple retrieval when it encounters trusted items and performs backward chaining over untrusted items. I will provide an evaluation of my proposed modifications that shows that my algorithm is conservative and that it provides significant gains in performance for certain queries.

6.1 Change Classification

The goal of this thesis is to design a hybrid reasoning architecture and develop a scalable reasoning system whose efficiency is able to meet the interaction requirements in a ScienceWeb system when facing a large knowledge base subject to changes. These changes could occur in the ontology, in the custom rule set, in instances already present in the knowledge base, or in the addition of new instances harvested from the web.

Changes to an ontology could include adding, removing or modifying classes or properties in the ontology. An example of modifying a property in an ontology would be a user changing the domain of the property "`publicationAuthor`" from class "`Publication`" to "`Article`". The "`publicationAuthor`" triples whose objects are not "`Article`" type will be removed for the consistency. After the changes, if someone were to then pose a query "Who are coauthors of Professor X?" , such a query could not

be answered without reasoning about the publication authors, which have been affected by these changes. By contrast, if someone were to pose a query "Who are the advisors of Student X?" , this query could be answered by direct lookup in the knowledge base. No reasoning would be required, and the answer would be unaffected by the changes.

Changes to custom rules include adding, removing or modifying definition of custom rules such as modifying the rule to define "`groundbreaking research`". An example of modifying a custom rule would be a user locating the existing qualitative descriptors for "groundbreaking professor" and changing the rule definition. After the change, now if someone were to pose the query "Who are coauthors of Professor X?", this query could be answered by direct lookup in the knowledge base. No reasoning would be required. By contrast, if the query were posed "Who are groundbreaking researchers in University X?", such a query could not be answered without reasoning about the implications of the changed rule.

Changes to instances include adding, removing or modifying instances in the storage, such as updating publication list of a professor. Table XXXIV shows the description of the changes to the ontology, instances and custom rules based on [184]. The column labeled "Operation Description" lists the descriptions of the operations.

6.2 A Trust-based Hybrid Reasoning Algorithm

Let us assume that the knowledge base contains all known and derived facts such that any query can be directly answered by retrieving the corresponding truths. In such an environment I now introduce a change and analyze the impact of that change. We consider three categories of changes: changes to the ontology, changes to custom rules and changes to instances.

TABLE XXXIV
DESCRIPTION OF THE CHANGES TO THE ONTOLOGY, INSTANCES AND CUSTOM RULES

	Operation Description
Ontology	Add a Class
	Remove a Class
	Modify a Class
	Add a Subclass
	Modify a Subclass
	Remove a Subclass
	Add a Superclass
	Modify a Superclass
	Remove a Superclass
	Add a EquivalentClass
	Remove a EquivalentClass
	Modify a EquivalentClass
	Add a Property
	Remove a Property
	Modify a property
	Add a Domain
	Remove a Domain
	Modify a Domain
	Add a Range
	Remove a Range
	Modify a Range
	Set Functionality
	Unset Functionality
	Add Symmetry
	Remove Symmetry
	Set Transitivity
	Unset Transitivity
	Set InverseFunctionality
	Unset InverseFunctionality
	Add a Superproperty
	Remove a Superproperty
	Modify a Superproperty
	Add a Subproperty
	Remove a Subproperty
	Modify a Subproperty
	Add an Equivalent Property
	Remove an Equivalent Property
	Modify an Equivalent Property
	Add an Inverse Property
	Remove an Inverse Property
	Modify an Inverse Property
	Change to a DatatypeProperty
	Change to a ObjectProperty
	Add an Individual
	Remove an Individual
	Add a SameAs Individual
	Remove a SameAs Individual
	Modify a SameAs Individual
	Add Type
	Modify Type

TABLE XXXIV (CONTINUED)

	Operation Description
Ontology	Remove Type
Instance	Add a Datatype Property instance
	Modify a Datatype property instance
	Remove a Datatype property instance
	Add an Object Property instance
	Modify an Object Property instance
	Remove an Object Property instance
Custom Rules	Add a custom rule
	Modify a custom rule
	Remove a custom rule

In a knowledge base that relies on materialization via forward-chaining, all derivable conclusions from known facts and rules are assumed to have been written into the knowledge base. This leads to fast responses to queries because all queries can be answered by direct search and retrieval. No reasoning is required at the time of the query.

The same queries could presumably be answered in a non-materialized knowledge base containing only harvested facts and rules via backward chaining from the properties mentioned in the query. This would typically be considerably slower than a direct lookup in a materialized knowledge base.

The drawback of relying on materialization is a loss of agility in responding to changes. Materialization of a large knowledge base is potentially time-consuming. It may require deferring queries for many hours or, alternatively, issuing results that are incomplete or incorrect.

Consider the following example. A student, `student0` has enrolled in a course, `Course0`. This piece of information has been discovered by a harvester and added to the knowledge base. Now if someone were to pose a query "Who is enrolled in `Course0`?", this query could be answered immediately by direct lookup in the knowledge base. No reasoning would be required. However let us also posit that the knowledge base already

contains the fact that `Prof0` teaches `Course0`. If a query were immediately posed "Who is being taught by `Prof0`?", such a query could not be answered without reasoning about the implications of a rule

```
enrolledIn(?Student,?Course?), teaches(?Faculty,?Course)
        :- isTaughtBy(?Student,?faculty)
```

But a search for materialized `isTaughtBy` instances in the knowledge base will fail to turn up the relationship between `Professor0` and the newly enrolled `Student0`, until the knowledge base is re-materialized to incorporate this and all other recent changes.

A hybrid reasoning algorithm can use backward chaining over "untrusted" portions of a knowledge base while using direct lookup to recover previously materialized conclusions from the "trusted" portion. Such an algorithm is shown in Fig. 14.

```
Substitutions prove (Goal g)
{
   if (g is trusted)
       retrieve Substitutions M from knowledge base by direct lookup of g;
       return M;
   else {
         Substitutions M = empty;
         for each rule R and Substitution σ1  such that
            the head of R σ1 matches g {
               M1  = proveTheRuleBody (R.body, σ1);
               if (M1 is not empty) // proof of rule succeeded
                   {
                     Substitutions M2 = all Substitutions in M1
                         for variables in the head of R;
                     M += M2;
                   }
             }
       }
    return M;
}
```

Fig. 9. A hybrid reasoning algorithm

In the algorithm shown in Fig. 14, R denotes a rule, consisting of a rule body (premises) and a rule head (conclusion), the rule head is true only when the rule body is true.

The `prove` function returns all substitutions for the variables in the goal for which that goal is provable. It does this by consulting each rule matching the goal and attempting to find substitutions satisfying the body of that rule. A key step is the test to see if the proof goal is "trusted." The results of this test determine whether we simply look up previously materialized instances or engage in a backward-chaining proof.

When we need to prove the rule body, we attempt to recursively prove each goal in the rule body one by one. The process of proving the rule body is shown in Fig. 15. In my actual implementation [182], I employed OLDT [170] and memorization to avoid deep recursion. The bindings from earlier goals would be substituted into the current goal for subsequent proof. After we prove the current goal, we join the new substitutions from that proof with the prior substitutions. In `joinSubstitutions`, we iterate over two sets of substitutions and compose every pair of substitutions in the cross product with common values.

```
bool, Substitutions proveTheRuleBody (body, Substitution)
{

  Substitutions M  = empty;
   for each goal g in body from Substitution {
       M1 = prove(g);
       if (M is empty)
          return false,empty;
       M = joinSubstitutions(M,M1);
   }
   return true, M;
}
```

Fig. 10. Process of proving the rule body

The advantage of such a hybrid algorithm is that it allows the knowledge base maintainers to defer expensive re-materializations for long periods of time (long, at least, compared to the frequency of changes) while still permitting accurate and timely responses to incoming queries.

6.3 Conservative Trust Assessment and Experiments

6.3.1 Conservative Trust Assessment

The preceding hybrid algorithm depends upon the idea of knowing when the currently materialized instances corresponding to a proof goal can be trusted to be correct and complete.

We will say that a proof goal `p(?X,?Y)` is *trustworthy* if all instances of that goal derivable from facts and rules in the knowledge base are present in that knowledge base as instances. Trustworthy goals can be safely resolved by direct lookup in the knowledge base.

In practice, we are unlikely to be able to identify precisely all goals that are trustworthy except by materializing the knowledge base, which, by definition, forces all goals to be trustworthy. We therefore seek less expensive options to approximate the set of trustworthy goals, a less expensive option for dividing the set of possible proof goals into trusted and untrusted sets.

A partition into trusted and untrusted sets is called *conservative* if no untrustworthy goals are trusted. If we are conservative in such an approximation to the collection of trustworthy goals, then my hybrid reasoner can be relied upon to give accurate responses.

An apparently plausible approach to a conservative trust rule would be *property-*

based trust: assume that any property P that was involved in a change (e.g., if a new instance `P(x0, y0)` was added to the knowledge base) is itself untrusted and then to take the closure of the "is used as a premise of" relation, that is, if an untrusted property Q occurs in the body of a rule used to prove R

```
…, Q(x,y), … :- R(w,z)
```

then R is also untrusted.

For example, suppose that a student, student0 has enrolled in a course, `Course0`, taught by `Prof0`. The addition of a new fact `enrolledIn(Student0,Course0)` to the knowledge base would cause the property `enrolledIn` to be untrusted. In addition, given the rule presented earlier deriving `isTaughtBy(?Student,?Faculty)` from (in part) `enrolledIn` instances, the property `isTaughtBy` would also untrusted.

The attraction of this definition of "untrusted" is that it requires analysis of only the rules in the knowledge base without consulting with the far more numerous instances. A knowledge base of many millions of triples might be expressed in terms of a few hundreds of properties and a comparable number of rules, making this definition of trust far easier to compute than the true trustworthy set.

Unfortunately, this simple procedure breaks down in the face of "meta-rules" in the knowledge base, rules that permit reasoning about properties themselves. For example, suppose that student, `Student0`, just got his degree from `University0`, and that the instance "`degreeFrom(student0, University0)`" is added to the knowledge base. We will posit that there are rules such that the property `degreeFrom` is an inverse property of `hasAlumnus`. The inverse rule implies that

```
?P(?X, ?Y), inverse(?P,?Q) :- ?Q(?Y,?X)
```

Immediately after adding the new fact to the knowledge base, queries such as "what alumni/alumnae does `university0` have?" would not respond with `Student0`. The `hasAlumnus` property is not trustworthy, but it would actually be left as trusted by the initial trust approximation. A more sophisticated definition of trust is required.

One possibility would be to expand the set of untrusted properties via special handling of the meta-rules common to RDF and OWL. As I will show, however, in my experimental results below, prototypes of such an expanded definition of property-based trust demonstrated that simple changes to a knowledge base could then result in significant fractions of the knowledge base being marked as untrusted. I concluded that properties do not offer a detailed enough discrimination to serve as a practical basis for trust.

I propose instead a concept of *pattern-based trust:* a pattern `P(X,Y)` (where `X` and `Y` could be ground instances or free variables) is untrusted if it matches a change to the knowledge base or if it can be derived from a rule with an untrusted pattern as a premise.

For example, suppose again that a student, `student0` has enrolled in a course, `Course0`, taught by `Prof0`. The addition of a new fact `enrolledIn(Student0,Course0)` to the knowledge base would cause the pattern `enrolledIn(Student0,Course0)` to be untrusted. In addition, given the rule presented earlier deriving `isTaughtBy(?Student,?Faculty)`, the pattern `isTaughtBy(Student0,Prof0)` would also be untrusted.

In my hybrid `prove` algorithm, presented in the prior section, the test to see if a goal `g` is trusted is now interpreted as "if `g` cannot be unified with any untrusted pattern". Hence queries and proof goals involving patterns such as `isTaughtBy(?S,Prof0)` and

`isTaughtBy(?S,?P)` would also be treated as untrusted.

Of importance is the fact that patterns (and therefore potential queries and proofs) involving other students and other faculty (e.g., "who is taught by `Prof1`?") remain trusted and so could be answered by direct lookup with no reasoning.

```
setOfPatterns propagateUntrustForward (premises, conclusion, substitutionSet,
existingUntrustedSet)
{
   untrustedSet = {};
   for each premise p  in premises
      {
       patternSet = {};
       goalList = get unified goals by replacing variables
          in p from substitutionSet;
       for each goal g in goalList
         {
           retrieve instances r from knowledge base by direct lookup of g;
           retrieve results r1 from realized patterns in
               existingUntrustedSet  by direct lookup of g;
           r += r1;
           if (r is empty)
              return empty;
           if (the size of r <threshold)
               untrustedpatternSet + = r;
            else if (the size of  r >=threshold)
               untrustedpatternSet + = g;
         }
        Substitutions M = unify p with untrustedpatternSet;
        substitutionSet =  joinSubstitutions(substitutionSet,M);
      }
  untrustedSet = substitute variables in conclusion with substitutionSet;
  return untrustedSet;
}
```

Fig. 11. The pattern-based trust marking algorithm

The pattern-based trust marking algorithm shown in Fig. 16 will work with these meta-rules as well as customized rules. We accumulate a set of already untrusted patterns

by running the collectUntrustedDueTo algorithm iteratively on each new change. The collectUntrustedDueTo function is shown in Fig. 17.

```
setOfPatterns collectUntrustedDueTo
     (oneChange, existingUntrustedSet)
{
   untrustedSet = {oneChange};
   for each rule R (p1^p2...^pi...^pn => q) and
      each pi matching oneChange
     {
        retrieve Substitutions M from knowledge base by
           direct lookup of pi;
        untrustSet+= propagateUntrustForward
           ([p0..pi-1,pi+1...pn],q, M}, existingUntrustedSet)
     }
   existingUntrustedSet += untrustSet;
   discard from existingUntrustedSet  any patterns
      that are specializations of other elements.
   return existingUntrustedSet;
}
```

Fig. 12. The collectUntrustedDueTo function

The collectUntrustedDueTo function collects untrusted patterns for one single change to the knowledge base, assuming we have already had an existing untrusted pattern set. The first time this algorithm runs after materialization that set will be empty. The one single change would be added to the untrusted set first. Then we check each rule in the rule set to see if we can propagate the "untrust" forward by a limited, specialized analogue of forward chaining. At last, we add our untrusted set produced from the above one change to the existing untrusted set, discarding any patterns that are specializations of other elements.

The propagateUntrustForward function goes through each premise of the

rule to compose and propagate the untrusted substitutions. The untrusted substitutions from the earlier solutions are substituted into the upcoming premise to yield multiple instances of that clause as goals for subsequent proof. When we prove a premise, we need to prove each unified goal produced from untrusted substitutions by replacing variables in the premise. Upon proving a unified goal, we retrieve matched instances from the knowledge base and `existingUntrustedSet` by a direct lookup of the unified goal.

I use a threshold to determine whether the actual matched results or the unified goal itself should be added to the untrusted substitutions. If the number of matched instances is comparatively large, I use a pattern that can represent the whole set of matched instances in the untrusted substitutions instead of the large set of matched instances itself. Including all the untrusted instances in the untrusted set would make it inefficient when we determine if a goal is trusted or not in this marking algorithm. Finally, the rule's conclusion (head), with appropriate substitution, is added to the untrusted set.

For example, consider a scenario, based on LUBM, in which a university, `University0,` has hired professor, `Fullprofessor0`. This piece of information `worksFor(Fullprofessor0, University0)` has been discovered by a harvester and added to the knowledge base. Given the OWL Horst rule set, according to my pattern-based marking algorithm, the untrusted patterns are:

```
worksFor (Fullprofessor0,  University0)
member (University0, Fullprofessor0)
memberOf (Fullprofessor0, University0)
```

Now if someone were to pose a query "Who are members of `University0`?", given that pattern `memberOf(Fullprofessor0, University0)`" is untrusted, we need to reason using backward chaining.

On the other hand, if someone were to pose a query "Who are members of

University1?", given that the pattern memberOf(?x, University1) is trusted, all the members of University1 can still be retrieved by a direct look up in the knowledge base.

6.3.2 Experiments

In this Section, I further explore the impact of trust rules on reasoning by conducting and presenting reports involving two sets of benchmarking experiments. I also describe preliminary experiments that explore how the percentage of trusted facts in the knowledge base affects the performance of the hybrid algorithm.

First, I compare the number of objects in the knowledge base marked as untrusted by my property-based algorithm to what should be really untrusted. This provides an indication of how many unnecessary reasoning steps the hybrid algorithm would have to go through.

TABLE XXXV
RESULTS FOR PROPERTY-BASED MARKING ALGORITHM

Changes	Actual # new properties	Actual # new facts	# untrusted properties
Adding a new class	2	3	12
Add a subclass relationship between two new classes	2	6	12
Add new Class as subClass of existing class	2	5	12
Adding a new Property	2	2	12
Add a new Property as subPropertyOf of another new Property	2	4	12
Add new Property as subPropertyOf of existing Property	2	3	12
Add new Class as domain to a new Property	3	5	13
Add new Class as range to a new Property	3	5	13

Table XXXV shows that property-based marking greatly exaggerates the number of untrusted properties. It compares the number of properties that should be

untrusted after a change vs. the numbers the property-based algorithm produces. Even in a knowledge base with many millions of triples, the number of distinct properties is likely to be counted in the tens to (very) low hundreds, so the increase shown there in the number of properties marked as untrusted would likely have a significant effect on query processing. This is made worse by the fact that the experiments showed that many properties marked as untrusted were ontological meta-rules such as all subclass relations. As an example, under LUBM(1) the properties marked as untrusted match an average of 97,300 triples out of 149,894, which is about 65% of the knowledge base. These results led us to set aside property-based trust marking in favor of the finer discrimination afforded by pattern-based marking.

In contrast the average number of patterns added by the pattern-based marking algorithm for the same changes as in produces the same number of properties as the 'actual' columns show.

The second set of experiments provides a comparison of performance of my hybrid pattern-based proof algorithm against the regular, optimized backward chaining algorithm [182] and against the OWLIM forward chaining algorithm using benchmark knowledge bases LUBM1, LUBM10, and LUBM40, of size 100,839, 1,272,871, and 5,307,754 objects respectively. Table XXXVI shows the comparison of response times for LUBM query 2 for these three algorithms after adding an existing student as a member of an existing department to the knowledge base.

Table XXXVII shows the comparison of response times for LUBM query 6 for these three algorithms after adding a new undergraduate student.

TABLE XXXVI
QUERY RESPONSE TIME (MS) AFTER ADDING STUDENT

	Hybrid	Backward	Forward (load +query)
LUBM1	93	490	960+3.4
LUBM10	546	1,060	7,800+150
LUBM40	2,548	9,100	350,000+5,100

TABLE XXXVII
QUERY RESPONSE TIME (MS) AFTER ADDING UNDERGRADUATE STUDENT

	Hybrid	Backward	Forward (load +query)
LUBM1	452	180	960+240
LUBM10	1,575	1,170	7,800+1,200
LUBM40	3,525	43,000	350,000+5,300

Both tables show that the hybrid algorithm, though slower on a query-by-query basis than forward chaining, is faster than backward chaining and at least an order of magnitude faster than re-materializing the knowledge base (which is the time that would be required to re-materialize a knowledge base after a change). I have performed preliminary experiments on determining the factors that affect the performance of the hybrid algorithm. I investigated the ratio of trusted to untrusted facts in the knowledge base after a set of changes and compared the performance of the three algorithms (backward chaining, forward chaining and hybrid) for different sizes of the knowledge base and selected queries. The following sample scenarios in Table XXXVIII illustrate the results of the preliminary experiments.

The percentage of untrusted facts in the knowledge base after executing the pattern-based marking algorithm ranges from close to 0 to a high of 10% in all experiments. The percentage of untrusted patterns ranges from close to zero to 5%. For

scenarios 1 and 2, the hybrid algorithm outperforms the backward chaining algorithm in terms of query response time significantly. For LUBM(50), the query response times in scenario 1 are 1,887ms and 3,229ms respectively. The query response times in scenario 2 are 9,937ms and 11,216ms, respectively. In scenario 3 the backward chaining algorithm outperforms the hybrid algorithm in terms of query response time (7,238ms and 5,054ms respectively). The reason for this is that, in scenario 3, the percentage of untrusted triples in the whole knowledge base is about 10% and the queries require resolving patterns mostly located in the untrusted list. For the queries selected, the hybrid algorithm outperforms the regular backward chaining algorithm by 30 percent on the average for the selected set of experiments.

TABLE XXXVIII
SAMPLE SCENARIOS

Scenario	Adding Changes	Query
Scenario 1	MiddleWork rdf:type rdfs:Class MiddleWork rdfs:subclassOf Work Course rdfs:subclassOf MiddleWork	?x rdf:type Work
Scenario 2	SupermemberOf rdf:type rdf:Property memberOf rdfs:subPropertyOf SupermemberOf	?x SupermemberOf ?y
Scenario 3	MiddledegreeFrom rdf:type rdf:Property MiddledegreeFrom rdfs: subPropertyOf degreeFrom undergraduateDegreeFromrdfs: subPropertyOf MiddledegreeFrom	?x degreeFrom ?y

The experiments reported here suggest that a hybrid reasoner based on trust can be effective on some moderately sized knowledge bases. In considering the likely scalability of this approach, we may consider scaling to both larger knowledge bases and to bases subject to more increasingly frequently change.

Because the pattern-based trust markup is similar in structure to forward chaining, as the size of the knowledge base increases, the time to assess trust should grow at a rate no higher than, and possibly lower than, the increase in time to re-instantiate. An important contributory factor will be the overall degree of inter-connection within the knowledge base semantics. A loosely connected network will lead to faster termination of the trust marking algorithm. The experiments reported here may actually understate the potential savings, as I expect that LUBM is more tightly inter-connected than many practical knowledge bases.

6.4 Evaluation of the Trust-based Hybrid Reasoning

In this section, I further explore how the percentage of untrusted facts in the knowledge base affects the performance of the hybrid algorithm, that is, explore the relationship between the percentage of untrusted facts in the knowledge base and query response time (clock time).

Experimental design

The ontology data consist of the benchmark knowledge bases LUBM (10) and LUBM (30), of size 2,240,657 and 6,449,543 triples respectively. I use as test queries the 14 queries from LUBM. Jena TDB has been adopted for the external storage of the knowledge base. The percentages of untrusted facts in the knowledge base are generated by making changes to the knowledge base. In this experiment, the changes to the knowledge base are removing different number of students in the knowledge base together with any impacted triples.

Results and discussion

Table XXXIX shows the number of students removed and the generated untrusted

percentage.

TABLE XXXIX
The Number of Students Removed and the Generated Untrusted Percentage

LUBM (10)	Untrusted Percentage	**8%**	**16%**	**24%**	**32%**	**40%**	**47%**	**54%**
	Number of Students	10,000	20,000	30,000	40,000	50,000	60,000	70,000
LUBM (30)	Untrusted Percentage	**5%**	**12%**	**16%**	**26%**	**31%**	**41%**	**55%**
	Number of Students	20,000	40,000	60,000	100,000	120,000	160,000	220,000

Table XL shows the response times (unit: ms) for the LUBM 14 queries as a function of different percentages of untrusted facts in the knowledge base LUBM (10).

TABLE XL
THE RESPONSE TIMES (MS) FOR THE LUBM 14 QUERIES WITH DIFFERENT PERCENTAGES OF UNTRUSTED FACTS IN KNOWLEDGE BASE LUBM (10)

	0%	**8%**	**16%**	**24%**	**32%**	**40%**	**47%**	**54%**
query1	775.	1412.	2138.	2594.	3217.	3409.	3911.	5128.
query2	1649.	3493.	4283.	5196.	5317.	5940.	7441.	7672.
query3	3.	272.	530.	805.	1061.	1307.	1531.	1999.
query4	31.	323.	593.	915.	1167.	2000.	1827.	2931.
query5	5.	385.	1115.	1259.	2109.	1881.	3164.	3986.
query6	404.	2133.	2739.	3483.	4047.	5238.	6374.	7326.
query7	1331.	2930.	3408.	4052.	4721.	5412.	6228.	7166.
query8	1616.	4181.	5006.	5555.	6342.	7513.	9336.	10243.
query9	640.	2621.	3585.	4372.	4805.	5381.	6464.	8453.
query10	79.	1845.	2029.	2793.	3292.	4415.	5799.	6087.
query11	9.	281.	574.	876.	1190.	1408.	1663.	2009.
query12	16.	296.	591.	933.	1410.	1446.	2130.	2338.
query13	3.	401.	731.	1270.	1662.	2288.	2435.	3004.
query14	158.	689.	1015.	1532.	1995.	2194.	3061.	3217.

In order to show the trend of query response time as the percentage of untrusted

facts change, I normalized the query response time to:

```
relativetime(x)  = (query response times x) / (query response time with 100%
trusted facts) where x is the % of untrusted facts.
```

Table XLI shows the variations of response times for LUBM 14 queries with different percentages of untrusted facts in knowledge base LUBM (10).

TABLE XLI
THE RELATIVE RESPONSE TIMES (MS) FOR THE LUBM 14 QUERIES WITH DIFFERENT PERCENTAGES OF UNTRUSTED FACTS IN KNOWLEDGE BASE LUBM (10)

	8%	16%	24%	32%	40%	47%	54%
query1	1.82	2.76	3.35	4.15	4.40	5.05	6.62
query2	2.118	2.597	3.151	3.224	3.60	4.512	4.652
query3	90	200	300	400	400	500	700
query4	10.	19.	30.	38.	60.	59.	94.
query5	80	200	200	400	400	600	800
query6	5.28	6.78	8.62	10.0	13.0	15.8	18.1
query7	2.201	2.560	3.044	3.547	4.066	4.679	5.384
query8	2.587	3.098	3.438	3.924	4.649	5.777	6.338
query9	4.10	5.60	6.83	7.51	8.41	10.1	13.2
query10	23.	26.	35.	42.	56.	73.	77.
query11	30	60	$1.*10^2$	100	200	200	200
query12	18.	37.	58.	88.	90.	130	150
query13	100	200	400	600	800	800	1000
query14	4.36	6.42	9.70	12.6	13.9	19.4	20.4

I have calculated the Pearson product-moment correlation coefficient between the query response time and the untrusted percentage. All the correlation coefficient values for the 14 queries are between 0.957 and 0.998. And the probability of significance p-value [185] (one-tailed) are all below significance level 0.01 (using Student's t

distribution), showing the correlation coefficient values are statistically significant. A strong and positive relationship between the query response time and the untrusted percentage are shown in Table XLI and confirmed by the above correlation coefficient values. I fit curves to the data in Table XLI using exponential regression for query 7. I present the trend lines with equation and coefficient of determination in Fig. 18. I fit curves to the data in Table XLI using linear regression for query 13. I present the trend lines with equation and coefficient of determination in of Fig. 19. The x-axis on the figures represents the untrusted percentage while y-axis represents the query response time.

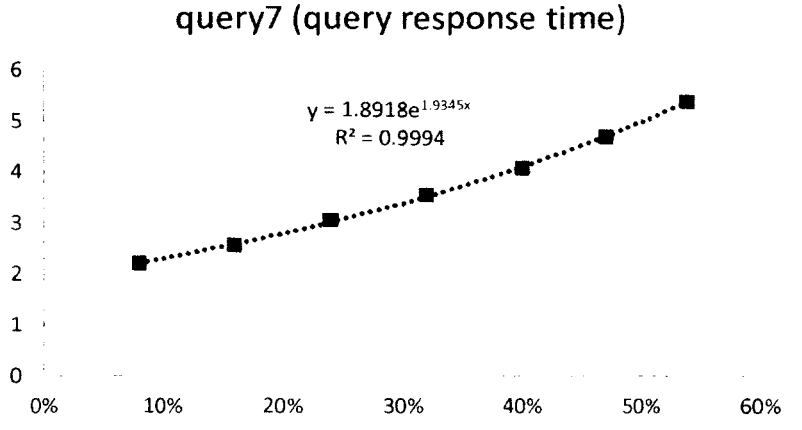


Fig. 18. An example of curve fitting using exponential regression for Query7 in LUBM (10)

Table XLII shows the Pearson product-moment correlation coefficients of linear, exponential and power curves for the data from Table XLI. As Table XLII shows, the difference among the liner, exponential and power correlation coefficients is between 0.0072 and 0.0469 for the 14 queries. There is no one correlation coefficient value that is

significantly larger than the others for each query. So we are not able to tell if the growth is linear or exponential or power due to the close correlation coefficient values.

TABLE XLII
THE CORRELATION COEFFICIENTS OF LINEAR, EXPONENTIAL AND POWER CURVE FOR QUERY RESPONSE TIME AND THE UNTRUSTED PERCENTAGE IN KNOWLEDGE BASE LUBM (10)

Query	Linear	Exponential	Power
query 1	0.9784	0.9769	0.9854
query 2	0.9796	0.9819	0.9695
query 3	0.9926	0.9730	0.9980
query 4	0.9596	0.9795	0.9835
query 5	0.9574	0.9461	0.9758
query 6	0.9884	0.9982	0.9713
query 7	0.9927	0.9997	0.9672
query 8	0.9794	0.9945	0.9476
query 9	0.9659	0.9848	0.9665
query 10	0.9768	0.9913	0.9461
query 11	0.9984	0.9624	0.9994
query 12	0.9875	0.9660	0.9949
query 13	0.9959	0.9679	0.9971
query 14	0.9892	0.9822	0.9894

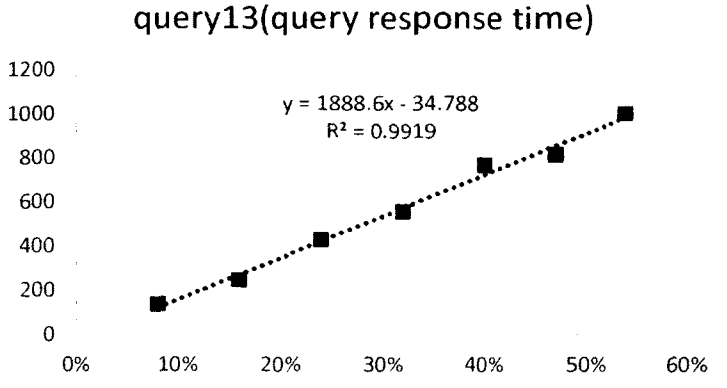


Fig. 19. An example of curve fitting using linear regression for Query13 in LUBM (10)

Table XLIII shows the response times for LUBM 14 queries with different percentages of untrusted facts using the knowledge base LUBM (30).In order to show the

trend of query response time as the percentage of untrusted facts change, I also did normalization to the original query response time. Table XLIV shows the normalized response times for the LUBM 14 queries with different percentages of untrusted facts in knowledge base LUBM (30).

TABLE XLIII
Response Times (ms) for LUBM 14 Queries with Different Percentages of Untrusted Facts in Knowledge Base LUBM (30)

	0%	5%	12%	16%	26%	31%	41%	55%
query1	4858.	6345.	4445.	5893.	10055.	12741.	14581.	20641.
query2	23622.	25975.	12576.	13446.	23669.	33979.	34361.	42458.
query3	25.	562.	1155.	1740.	3031.	3565.	7939.	9416.
query4	546.	935.	1365.	3116.	4479.	6021.	7384.	11848.
query5	16.	887.	1534.	2471.	5206.	6714.	8627.	11781.
query6	5659.	12074.	7867.	10156.	15302.	21988.	23184.	32498.
query7	8216.	9559.	10093.	11942.	15823.	20997.	24054.	30402.
query8	9590.	13380.	15244.	16914.	21406.	23186.	28721.	37282.
query9	6073.	11687.	11542.	15634.	17752.	21313.	24462.	32695.
query10	290.	3991.	6093.	8074.	10901.	13405.	17355.	22698.
query11	312.	704.	1272.	1984.	4002.	5052.	5034.	7881.
query12	270.	635.	1271.	2651.	3168.	4006.	6123.	8637.
query13	16.	852.	1723.	2652.	4960.	6249.	7135.	13212.
query14	1562.	1763.	2593.	4277.	4991.	6143.	8060.	10747.

I have again calculated the Pearson product-moment correlation coefficient between the query response time and the untrusted percentage. All the correlation coefficient value for the 14 queries are between 0.818 and 0.998, and 13 probability of significance p-values [185] (one-tailed) are all below significance level 0.01 (using Student's t distribution), showing all these 13 correlation coefficient values are statistically significant except for one. Only one probability of significance p-value (one-tailed) is 0.012, which is a little greater than significance level 0.01, but still less than

0.05. A strong and positive relationship between the query response time and the untrusted percentage are shown in Table XLIV and confirmed by the above correlation coefficient values. I fit curves to the data in Table XLIV using exponential regression for query 7. I present the trend lines with equation and coefficient of determination in Fig. 20. I fit curves to the data in Table XLIV using linear regression for query 13. I present the trend lines with equation and coefficient of determination in of Fig. 21.

TABLE XLIV
THE RELATIVE RESPONSE TIMES FOR LUBM 14 QUERIES WITH DIFFERENT PERCENTAGES OF UNTRUSTED FACTS IN KNOWLEDGE BASE LUBM (30)

	0.05	**0.12**	**0.16**	**0.26**	**0.31**	**0.41**	**0.55**
query1	1.306	0.9150	1.213	2.070	2.623	3.001	4.249
query2	1.0996	0.53238	0.56922	1.0020	1.4384	1.4546	1.7974
query3	22.	46.	70.	120	140	320	380
query4	1.71	2.50	5.71	8.20	11.0	13.5	21.7
query5	55.	96.	150	320	420	540	740
query6	2.134	1.390	1.795	2.704	3.885	4.097	5.743
query7	1.163	1.228	1.454	1.926	2.556	2.928	3.700
query8	1.395	1.590	1.764	2.232	2.418	2.995	3.888
query9	1.924	1.900	2.574	2.923	3.509	4.028	5.384
query10	13.8	21.0	27.8	37.6	46.2	59.8	78.3
query11	2.26	4.08	6.36	12.8	16.2	16.1	25.2
query12	2.35	4.71	9.82	11.7	14.8	22.7	32.0
query13	53	110	160	310	390	440	820
query14	1.129	1.660	2.738	3.195	3.933	5.16	6.880

Table XLV shows the Pearson product-moment correlation coefficients of linear, exponential and power curves for the data from Table XLIV. As Table XLV shows, the difference among the liner, exponential and power correlation coefficients is between 0.0133 and 0.1720 for the 14 queries. There is no one correlation coefficient value that is

significantly larger than the others for each query. So we are not able to tell if the growth is linear or exponential or power due to the close correlation coefficient values.

TABLE XLV
The Correlation Coefficients of Linear, Exponential and Power Curve for Query Response Time and the Untrusted Percentage in Knowledge Base LUBM (30)

Query	Linear	Exponential	Power
query 1	0.9666	0.9357	0.8411
query 2	0.8176	0.7452	0.5732
query 3	0.9695	0.9700	0.9833
query 4	0.9887	0.9516	0.9778
query 5	0.9959	0.9529	0.9869
query 6	0.9451	0.9107	0.7941
query 7	0.9883	0.9820	0.9380
query 8	0.9925	0.9990	0.9471
query 9	0.9857	0.9825	0.9296
query 10	0.9986	0.9747	0.9900
query 11	0.9822	0.9363	0.9863
query 12	0.9889	0.9447	0.9875
query 13	0.9791	0.9608	0.9916
query 14	0.9935	0.9608	0.9842

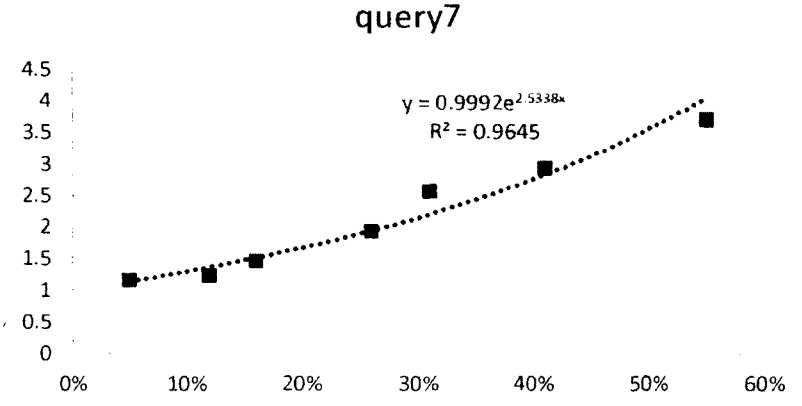


Fig. 20. An example of curve fitting using exponential regression for Query7 in LUBM (30)

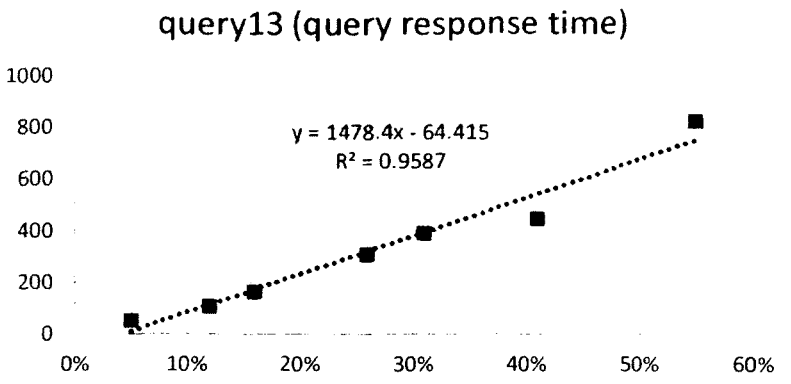


Fig. 21. An example of curve fitting using linear regression for Query13 in LUBM (30)

Based on these experiments, there is a positive relationship between the query response time and the untrusted percentage. however the rate of growth is never much worse than linear for untrusted percentages below 50%, indicating that the data point at which we need to run forward chaining to improve the query response time has not explicitly appeared before half of the knowledge base is untrusted, and that the trust-based hybrid reasoning algorithm works fine when the untrusted percentages are below 50%. I was actually expecting a faster-than-linear growth to help me to find the point at which we need to run forward chaining. However, I have not observed this faster-than-linear growth in this experiment. The nearly-linear behavior observed in these experiments suggests that the hybrid reasoner's performance may degrade only smoothly and robustly after substantial numbers of changes.

CHAPTER 7

CONTRIBUTIONS AND FUTURE WORK

7.1 Conclusions

ScienceWeb is a system that collects information about a research community and allows users to ask qualitative and quantitative questions related to that information using a reasoning engine. The more complete the knowledge base is, the more helpful answers the system will provide. As the size of knowledge base increases, scalability becomes a challenge for the reasoning system. It may handle millions even hundreds of millions of items in the knowledge base. As users make changes to the knowledge base and/or new information is collected, providing fast enough response time (ranging from seconds to a few minutes) is one of the core challenges for the reasoning system.

In this thesis I researched the issues involved in designing a hybrid reasoning architecture and developing a scalable reasoning system whose scalability and efficiency are able to meet the requirements of query and answering in a semantic web system when facing both a fixed knowledge base and an evolving knowledge base.

The objectives of my thesis were:

- Support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing a fixed knowledge base
- Support scalable reasoning of ScienceWeb to answer qualitative questions effectively when facing an evolving knowledge base

My research has met these objectives. Interposing a backward chaining reasoner

between an evolving knowledge base and a query manager with support of "trust" yields an architecture that can support reasoning in the face of frequent changes. An optimized query-answering algorithm, an optimized backward chaining algorithm and a trust-based hybrid reasoning algorithm are three key algorithms in such an architecture. I described these three algorithms and the corresponding evaluations in Chapter 4, 5 and 6 respectively. Collectively, these three algorithms are significant contributions to the field of backward chaining reasoners over ontologies.

When comparing to a traditional backward-chaining reasoner, the implementation of the optimized query answering algorithm is better in: 1) handling much larger knowledge base; 2) working with more complete rule sets (including all of the OWL rules); 3) responding to queries significantly faster in most cases.

When comparing the results with and without optimization techniques, the optimization techniques improved the efficiency of the backward chaining algorithm significantly in terms of time and space when compared to standard backward-chaining reasoners. When comparing the results with the forward-chaining reasoner in scenarios where the knowledge base is subject to frequent change, the optimized algorithm outperformed the forward-chaining reasoner.

When comparing the performance of a forward chaining algorithm to that of a pure backward chaining algorithm, the trust-based hybrid reasoning algorithm is better in almost all the cases tested.

With the support of rule-based reasoning, the ScienceWeb is able to answer qualitative questions. With the support of external storage, I extended our study to a knowledge base of more than 10 Million triples, increasing the scalability of the

backward chaining reasoning system. With the concept of "trust" and optimization techniques, I increased the efficiency of the reasoning system that we have proposed. In short, I have designed a hybrid reasoning architecture and developed a scalable reasoning system whose scalability and efficiency are able to meet the requirements of query and answering in a semantic web system when facing both a fixed knowledge base and an evolving knowledge base.

7.2 Future Work

In the future I will attempt to scale the knowledge base to the billion-triple level using further algorithm optimization and will design a new external storage management system.

I have not yet explored the impact of long sequences of individual changes on the marking algorithm time nor subsequently on the hybrid reasoner. In future work, I plan to explore the performance of the trust marking algorithm and of the hybrid reasoner as a function of the fraction of the knowledge base that is untrusted, a measure that would combine both the number of changes and the extent of their impact throughout the semantic graph.

Finally, I would like to explore the dynamic aspects of the queries over time and to assess the impact of the distribution of the queries and of the number of changes and the type of changes in the same period.

REFERENCES

[1] Microsoft. "*Microsoft Academic Search*," March 18, 2014 [online] Available: http://academic.research.microsoft.com/.

[2] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma, "Object-level ranking: bringing order to web objects," in *Proc. WWW '05*, Chiba, Japan, 2005, pp. 567-574.

[3] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen, "Community Information Management," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* vol. 29, no. 1, pp. 64-72, March, 2006.

[4] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *Proc. KDD '08*, Las Vegas, NV, USA, 2008, pp. 990-998.

[5] H. Shi, K. Maly, S. Zeil, and M. Zubair, "Comparison of Ontology Reasoning Systems Using Custom Rules," in *Proc. WIMS '11*, Sogndal, Norway, 2011.

[6] A. Yaseen, K. Maly, S. J. Zeil, and M. Zubair, "Performance Evaluation of Oracle Semantic Technologies with Respect to User Defined Rules," in *Proc. DEXA2011*, Toulouse, France, 2011, pp. 252-256.

[7] K. Maly, S. Zeil, and M. Zubair. "*ScienceWeb pre-survey*," December 3, 2010 [online] Available: http://www.cs.odu.edu/~zeil/scienceWeb/survey10/.

[8] S. J. Russell, and P. Norvig, *Artificial intelligence: a modern approach*, 1st ed., Upper Saddle River, New Jersey: Prentice hall, 1995.

[9] IEEE. "*IEEE-The world's largest professional association for the advancement of technology*," March 18, 2014 [online] Available: http://www.ieee.org/index.html.

[10] ACM Inc. "*ACM Digital Library*," March 18, 2014 [online] Available: http://portal.acm.org/.

[11] Springer. "*Springer-International Publisher Science,Technology,Medicine*," March 18, 2014 [online] Available: http://www.springer.com.

[12] Google. "*Google Scholar*," March 18, 2014 [online] Available: http://scholar.google.com/.

[13] M. Ley. "*DBLP Bibliography*," March 18, 2014 [online] Available: http://www.informatik.uni-trier.de/~ley/db/.

[14] M. Ley, "The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives," *String Processing and Information Retrieval*, A. H. F. Laender and A. L. Oliveira, eds., pp. 1-10, Heidelberg Germany: Springer, 2002.

[15] The Pennsylvania State University. "*CiteSeerX*," March 18, 2014 [online] Available: http://citeseerx.ist.psu.edu/.

[16] C. L. Giles, K. D. Bollacker, and S. Lawrence, "CiteSeer: An automatic citation indexing system," in *Proc. DL '98*, Pittsburgh, PA, USA, 1998, pp. 89-98.

[17] getCITED Inc. "*getCITED:Academic Research,citation reports and discussion lists*," March 18, 2014 [online] Available: http://www.getcited.org/.

[18] H. Bast, A. Chitea, F. Suchanek, and I. Weber, "ESTER: Efficient Search on Text, Entities, and Relations," in *Proc. SIGIR '07*, Amsterdam, The Netherlands, 2007, pp. 671–678.

[19] M. J. Cafarella, C. Re, D. Suciu, O. Etzioni, and M. Banko, "Structured querying of web text," in *Proc. CIDR2007*, Asilomar, California, USA, 2007, pp. 225–234.

[20] T. Cheng, X. Yan, and K. C.-C. Chang, "EntityRank: searching entities directly and holistically," in *Proc. VLDB '07*, Vienna,Austria, 2007, pp. 387-398.

[21] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan, "Building structured web community portals: A top-down, compositional, and incremental approach," in *Proc. VLDB '07*, Vienna, Austria, 2007, pp. 399-410.

[22] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum, "Naga: Searching and ranking knowledge," in *Proc. ICDE2008*, Cancun, Mexico, 2008, pp. 953–962.

[23] D. N. Milne, I. H. Witten, and D. M. Nichols, "A knowledge-based search engine powered by wikipedia," in *Proc. CIKM '07*, Lisbon, Portugal 2007, pp. 445-454.

[24] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan, "DBLife: A community information management platform for the database research community," in *Proc. CIDR2007*, Asilomar, CA, USA, 2007, pp. 169-172.

[25] Department of Mathematics North Dakota State University. "*The Mathematics Genealogy Project*," March 18, 2014 [online] Available: http://genealogy.math.ndsu.nodak.edu/.

[26] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia-A crystallization point for the Web of Data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 3, pp. 154-165, September, 2009.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A large ontology from wikipedia and wordnet," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 6, no. 3, pp. 203-217, September, 2008.

[28] B. Aleman-Meza, F. Hakimpour, I. B. Arpinar, and A. P. Sheth, "SwetoDblp ontology of Computer Science publications," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, no. 3, pp. 151-155, September, 2007.

[29] H. Glaser, I. C. Millard, and A. Jaffri, "Rkbexplorer. com: a knowledge driven infrastructure for linked data providers," *The Semantic Web: Research and Applications*, S. Bechhofer, M. Hauswirth, J. Hoffmann and M. Koubarakis, eds., pp. 797-801, Heidelberg, Germany: Springer, 2008.

[30] Wikipedia. "*Wiki-Wikipedia,the free encyclopedia*," March 18, 2014 [online] Available: http://en.wikipedia.org/wiki/Wiki.

[31] Princeton University. "*About WordNet*," March 18, 2014 [online] Available: http://wordnet.princeton.edu/.

[32] I. Niskanen, and J. Kantorovitch, "Ontology driven data mining and information visualization for the networked home," in *Proc. RCIS2010*, Nice, France 2010, pp. 147-156.

[33] M. Zeman, M. Ralbovsky, V. Svátek, and J. Rauch, "Ontology-Driven Data Preparation for Association Mining," in *Proc. Znalosti2009*, Brno, Czech Republic, 2009, pp. 270-283.

[34] Y.-T. Kuo, A. Lonie, L. Sonenberg, and K. Paizis, "Domain ontology driven data mining: a medical case study," in *Proc. DDDM '07*, San Jose, CA, USA, 2007, pp. 11-17.

[35] S. Singh, P. Vajirkar, and Y. Lee, "Context-aware Data Mining using Ontologies," *Conceptual Modeling - ER 2003*, I.-Y. Song, S. W. Liddle, T.-W. Ling and P. Scheuermann, eds., pp. 405-418, Heidelberg, Germany: Springer, 2003.

[36] H.-J. Happel, and a. S. Seedorf, "Applications of ontologies in software engineering," in *Proc. SWESE2006*, Athens,GA, USA, 2006.

[37] V. Lopez, M. Pasin, and E. Motta, "Aqualog: An ontology-portable question answering system for the semantic web," *The Semantic Web: Research and Applications*, A. Gómez-Pérez and J. Euzenat, eds., pp. 546-562, Heidelberg, Germany: Springer, 2005.

[38] V. Tablan, D. Damljanovic, and K. Bontcheva, "A natural language query interface to structured information," *The Semantic Web: Research and Applications*, S. Bechhofer, M. Hauswirth, J. Hoffmann and M. Koubarakis, eds., pp. 361-375, Heidelberg, Germany: Springer, 2008.

[39] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl, "From keywords to semantic queries--Incremental query construction on the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 7, no. 3, pp. 166-176, September, 2009.

[40] A. Diaz, and G. Baldo, "Co-protégé: A groupware tool for supporting collaborative ontology design with divergence," in *Proc. Eighth International Protégé Conference* Madrid, Spain, 2005, pp. 32-32.

[41] T. R. Gruber, *Ontolingua: A mechanism to support portable ontologies*, Technical Report KSL-91-66. Knowledge Systems Laboratory, Stanford University, Stanford, California, 1992.

[42] B. Swartout, R. Patil, K. Knight, and T. Russ, "Ontosaurus: a tool for browsing and editing ontologies," in *Proc. KAW96*, Banff, Canada, 1996.

[43] Y. Sure, J. Angele, and S. Staab, "OntoEdit: Guiding ontology development by methodology and inferencing," *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, R. Meersman and Z. Tari, eds., pp. 1205-1222, Heidelberg, Germany: Springer, 2010.

[44] Ó. Corcho, M. Fernández-López, A. Gómez-Pérez, and Ó. Vicente, "WebODE: An integrated workbench for ontology representation, reasoning, and exchange," *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, A. Gómez-Pérez and V. R. Benjamins, eds., pp. 295-310, Heidelberg, Germany: Springer, 2002.

[45] R. Volz, D. Oberle, S. Staab, and B. Motik, "Kaon server-a semantic web management system," in *Proc. WWW2003*, Budapest, Hungary, 2003, pp. 20-24.

[46] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, "OilEd: a reason-able ontology editor for the semantic web," *KI 2001: Advances in Artificial Intelligence*, F. Baader, G. Brewka and T. Eiter, eds., pp. 396-408, Heidelberg, Germany: Springer, 2001.

[47] T. Jie, Z. Jing, Y. Limin, L. Juanzi, Z. Li, and S. Zhong, "ArnetMiner: extraction and mining of academic social networks," in *Proc. Proceeding of the 14th ACM*

SIGKDD international conference on Knowledge discovery and data mining, Las Vegas, Nevada, USA, 2008.

[48] W3C. "*W3C Semantic Web Activity*," March,18, 2014 [online] Available: http://www.w3.org/2001/sw/.

[49] M. Baziz, M. Boughanem, and S. Traboulsi, "A concept-based approach for indexing documents in IR," in *Proc. INFORSID2005*, Grenoble, France, 2005, pp. 489-504.

[50] L. Khan, D. McLeod, and E. Hovy, "Retrieval effectiveness of an ontology-based model for information selection," *The VLDB Journal—The International Journal on Very Large Data Bases,* vol. 13, no. 1, pp. 71-85, January, 2004.

[51] H. Mosteghanemi, and H. Drias, "Bees Swarm Optimization for Real Time Ontology Based Information Retrieval," in *Proc. WI-IAT '12*, Macau, China, 2012, pp. 154-158.

[52] R. Mihalcea, and D. Moldovan, "Semantic indexing using WordNet senses," in *Proc. ACL '2000*, Hong Kong, China, 2000, pp. 35-45.

[53] S. Kara, Ö. Alan, O. Sabuncu, S. Akpınar, N. K. Cicekli, and F. N. Alpaslan, "An ontology-based retrieval system using semantic indexing," *Information Systems,* vol. 37, no. 4, pp. 294-305, June, 2012.

[54] D. Vallet, M. Fernández, and P. Castells, "An ontology-based information retrieval model," *The Semantic Web: Research and Applications*, A. Gómez-Pérez and J. Euzenat, eds., pp. 455-470, Heidelberg, Germany: Springer, 2005.

[55] A. Ferrara, L. A. Ludovico, S. Montanelli, S. Castano, and G. Haus, "A semantic web ontology for context-based classification and retrieval of music resources,"

ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), vol. 2, no. 3, pp. 177-198, August, 2006.

[56] A. Maedche, G. Neumann, and S. Staab, "Bootstrapping an ontology-based information extraction system," *Intelligent exploration of the web*, P. S. Szczepaniak, J. Segovia and L. A. Zadeh, eds., pp. 345-359, Heidelberg, Germany: Physica-Verlag GmbH, 2003.

[57] A. Bawakid, and M. Oussalah, "A semantic-based text classification system," in *Proc. CIS2010*, Reading, UK, 2010, pp. 1-6.

[58] P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein, "Applying semantic web services to bioinformatics: Experiences gained, lessons learnt," *The Semantic Web–ISWC 2004*, S. A. McIlraith, D. Plexousakis and F. v. Harmelen, eds., pp. 350-364, Heidelberg, Germany: Springer, 2004.

[59] K. Wolstencroft, A. Brass, I. Horrocks, P. Lord, U. Sattler, D. Turi, and R. Stevens, "A little semantic web goes a long way in biology," *The Semantic Web–ISWC 2005*, Y. Gil, E. Motta, V. R. Benjamins and M. A. Musen, eds., pp. 786-800, Heidelberg, Germany: Springer, 2005.

[60] N. Barnickel, J. Böttcher, and A. Paschke, "Incorporating semantic bridges into information flow of cross-organizational business process models," in *Proc. I-SEMANTICS '10*, Graz, Austria, 2010.

[61] J. Davies, D. Fensel, and F. v. Harmelen, *Towards the Semantic Web: Ontology-driven Knowledge Management*, 1st ed., Hoboken, NJ: Wiley, 2003.

[62] D. Brickley, and L. Miller. "*The Friend of a Friend (FOAF) project*," March 18, 2014 [online] Available: http://www.foaf-project.org/.

[63] J. G. Breslin, A. Harth, U. Bojars, and S. Decker, "Towards semantically-interlinked online communities," *The Semantic Web: Research and Applications*, A. Gómez-Pérez and J. Euzenat, eds., pp. 500-514, Heidelberg, Germany: Springer, 2005.

[64] A. Doms, and M. Schroeder, "GoPubMed: exploring PubMed with the gene ontology," *Nucleic acids research*, vol. 33, no. suppl 2, pp. W783-W786, 2005.

[65] NIH. "*PubMed Central*," March 18, 2014 [online] Available: http://www.ncbi.nlm.nih.gov/pmc/.

[66] National Academy of Sciences. "*The National Academies: Advisers to the Nation on Science, Engineering, and Medicine*," March 18, 2014 [online] Available: http://www.nationalacademies.org/.

[67] D. Nardi, and R. J. Brachman, "An introduction to description logics," *The description logic handbook: theory, implementation, and applications*, F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, eds., pp. 5–44, New York, NY: Cambridge University Press, 2002.

[68] M. Minsky, "A framework for representing knowledge," *The Psychology of Computer Vision*, pp. 211-277, 1975.

[69] M. R. Quillian, "Semantic memory," *Semantic information processing*, M. Minsky, ed., pp. 227–270, Cambridge, MA: MIT press, 1968.

[70] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, *The description logic handbook: theory, implementation, and applications*, 2nd ed., New York, NY: Cambridge University Press, 2007.

[71] F. Baader, I. Horrocks, and U. Sattler, "Description logics," *Foundations of Artificial Intelligence. Handbook of Knowledge Representation*, F. v. Harmelen, V. Lifschitz and B. Porter, eds., pp. 135-179, Amsterdam: Elsevier, 2008.

[72] Wikipedia. "*Discription Logic*," March 18, 2014 [online] Available: http://en.wikipedia.org/wiki/Description_logic.

[73] D. Nardi, and R. J. Brachman, "An Introduction to Description Logics," *The description logic handbook* F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, eds., pp. 1-40, New York, NY, USA: Cambridge University Press, 2003.

[74] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition - Special issue: Current issues in knowledge modeling* vol. 5, no. 2, pp. 199-220, June, 1993.

[75] E. Mays, R. Dionne, and R. Weida, "K-Rep system overview," *ACM SIGART Bulletin,* vol. 2, no. 3, pp. 93-97, June, 1991.

[76] C. Areces, H. D. Nivelle, and M. D. Rijke, "Resolution in modal, description and hybrid logic," *Journal of Logic and Computation,* vol. 11, no. 5, pp. 717-736, 2001.

[77] U. Hustadt, B. Motik, and U. Sattler, "Reducing SHIQ- description logic to disjunctive datalog programs," in *Proc. KR2004*, Morgan Kaufmann, Los Altos, 2004, pp. 152–162.

[78] U. Hustadt, and R. A. Schmidt, "On the relation of resolution and tableaux proof systems for description logics," in *Proc. IJCAI '99*, Stockholm, Sweden, 1999, pp. 110-117.

[79] U. Hustadt, and R. A. Schmidt, "Issues of decidability for description logics in the framework of resolution," *Automated Deduction in Classical and Non-Classical Logics*, R. Caferra and G. Salzer, eds., pp. 191-205, Heidelberg, Germany: Springer, 2000.

[80] Y. Kazakov, and B. Motik, "A Resolution-Based Decision Procedure for SHOIQ," *Journal of Automated Reasoning,* vol. 40, no. 2-3, pp. 89-116, March, 2008.

[81] F. Baader, J. Hladik, and C. Lutz, "From tableaux to automata for description logics," *Fundamenta Informaticae,* vol. 57, pp. 1-33, 2003.

[82] C. Lutz, "Interval-based temporal reasoning with general TBoxes," in *Proc. IJCAI '01*, Seattle, Washington, USA, 2001, pp. 85-96.

[83] C. Lutz, and U. Sattler, "Mary likes all cats," in *Proc. DL2000*, Aachen, Germany, 2000, pp. 213-226.

[84] F. Baader, and U. Sattler, "An overview of tableau algorithms for description logics," *Studia Logica,* vol. 69, no. 1, pp. 5-40, October, 2001.

[85] M. Fitting, "Tableau methods of proof for modal logics," *Notre Dame Journal of Formal Logic,* vol. 13, no. 2, pp. 237-247, 1972.

[86] M. Schmidt-Schaubß, and G. Smolka, "Attributive concept descriptions with complements," *Artificial intelligence,* vol. 48, no. 1, pp. 1-26, February, 1991.

[87] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," in *Proc. KR2006*, Lake District, UK, 2006, pp. 57–67.

[88] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial intelligence,* vol. 19, no. 1, pp. 17-37, September, 1982.

[89] A. Kiryakov, D. Ognyanov, and D. Manov, "OWLIM–a pragmatic semantic repository for OWL," *Web Information Systems Engineering - WISE 2005 Workshops*, M. Dean, Y. Guo, W. Jun, R. Kaschek, S. Krishnaswamy, Z. Pan and Q. Z. Sheng, eds., pp. 182-192, Heidelberg, Germany: Springer, 2005.

[90] Ontotext. "*OWLIM-SE*," March 18, 2014 [online] Available: http://owlim.ontotext.com/display/OWLIMv43/OWLIM-SE

[91] Oracle Corporation. "*Oracle Database 11g R2* " March 18, 2014 [online] Available: http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/

[92] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan, "Minerva: A scalable OWL ontology storage and inference system," *The Semantic Web–ASWC 2006*, R. Mizoguchi, Z. Shi and F. Giunchiglia, eds., pp. 429-443, Heidelberg, Germany: Springer, 2006.

[93] Z. Pan, and J. Heflin, "DLDB: Extending Relational Databases to Support Semantic Web Queries," in *Proc. ISWC2003*, Sanibel Island, Florida, USA, 2003.

[94] O. Erling, and I. Mikhailov, "RDF Support in the Virtuoso DBMS," *Networked Knowledge-Networked Media*, T. Pellegrini, S. Auer, K. Tochtermann and S. Schaffert, eds., pp. 7-24, Heidelberg, Germany: Springer, 2009.

[95] Franz Inc. "*AllegroGraph RDFStore Web 3.0's Database*," March 18, 2014 [online] Available: http://www.franz.com/agraph/allegrograph/.

[96] J. Dolby, A. Fokoue, A. Kalyanpur, E. Schonberg, and K. Srinivas, "Scalable highly expressive reasoner (SHER)," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 7, no. 4, pp. 357-361, December, 2009.

[97] Y. Chen, J. Ou, Y. Jiang, and X. Meng, "HStar-a semantic repository for large scale OWL documents," *The Semantic Web–ASWC 2006*, R. Mizoguchi, Z. Shi and F. Giunchiglia, eds., pp. 415-428, Heidelberg, Germany: Springer, 2006.

[98] The Apache Software Foundation. "*Apache Jena*," March 18, 2014 [online] Available: http://jena.apache.org.

[99] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American,* vol. 284, no. 5, pp. 34-43, May, 2001.

[100] G. Antoniou, C. V. Damasio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, and P. F. Patel-Schneider, *Combining Rules and Ontologies-A survey*, FP6 NoE REWERSE, Deliverable I3-D3., 2005.

[101] J. Mei, Z. Lin, and H. Boley, "ALC: an integration of description logic and general rules," in *Proc. RR '07*, Innsbruck, Austria, 2007, pp. 163-177.

[102] R. Rosati, "Semantic and computational advantages of the safe integration of ontologies and rules," *Principles and Practice of Semantic Web Reasoning*, F. Fages and S. Soliman, eds., pp. 50-64, Heidelberg, Germany: Springer, 2005.

[103] R. Rosati, "DL+ log: Tight integration of description logics and disjunctive datalog," in *Proc. KR2006*, Lake District of the United Kingdom, 2006, pp. 68–78.

[104] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. "*SWRL: A semantic web rule language combining OWL and RuleML, W3C Member submission, World Wide Web Consortium*," March 18, 2014 [online] Available: http://www.w3.org/Submission/SWRL/.

[105] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: Combining logic programs with description logic," in *Proc. WWW2003*, Budapest, Hungary, 2003, pp. 48-57.

[106] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, "AL-log: Integrating datalog and description logics," *Journal of Intelligent Information Systems,* vol. 10, no. 3, pp. 227-252, May, 1998.

[107] R. Rosati, "Towards expressive KR systems integrating datalog and description logics: Preliminary report," in *Proc. DL'99*, Linköping, Sweden, 1999, pp. 160–164.

[108] W. Drabent, J. Henriksson, and J. Małuszyński, "HD-rules: a hybrid system interfacing Prolog with DL-reasoners," in *Proc. ALPSWS2007*, Porto, Portugal 2007, pp. 76-90.

[109] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits, "NLP-DL: A Knowledge-Representation System for Coupling Nonmonotonic Logic Programs with Description Logics," in *Proc. ISWC2005*, Galway, Ireland, 2005.

[110] A. Y. Levy, and M.-C. Rousset, "Combining Horn rules and description logics in CARIN," *Artificial intelligence,* vol. 104, no. 1-2, pp. 165-209, September, 1998.

[111] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits, "Combining answer set programming with description logics for the semantic web," *Artificial intelligence,* vol. 172, no. 12-13, pp. 1495-1539, August, 2008.

[112] R. Rosati, "On the decidability and complexity of integrating ontologies and rules," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 3, no. 1, pp. 61-73, July, 2005.

[113] J. Mei, H. Boley, J. Li, V. C. Bhavsar, and Z. Lin, "DatalogDL: Datalog Rules Parameterized by Description Logics," *Canadian Semantic Web*, M. T. Koné and D. Lemire, eds., pp. 171-187, New York, NY, USA: Springer, 2010.

[114] J. Grant, and D. Beckett. "*RDF test cases*," March 18, 2014 [online] Available: http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/.

[115] P. Hayes, and B. McBride. "*RDF semantics*," March 18, 2014 [online] Available: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[116] H. J. t. Horst, "Combining RDF and part of OWL with rules: Semantics, decidability, complexity," *The Semantic Web - ISWC2005*, Y. Gil, E. Motta, V. R. Benjamins and M. A. Musen, eds., pp. 668-684, Heidelberg Germany: Springer, 2005.

[117] H. J. t. Horst, "Extending the RDFS entailment lemma," *The Semantic Web–ISWC 2004*, S. A. McIlraith, D. Plexousakis and F. v. Harmelen, eds., pp. 77-91, Heidelberg, Germany: Springer, 2004.

[118] H. J. t. Horst, "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary," *Web*

Semantics: Science, Services and Agents on the World Wide Web, vol. 3, no. 2-3, pp. 79-115, October, 2005.

[119] V. Haarslev, and R. Möller, "Racer: A core inference engine for the semantic web," in *Proc. EON2003*, Sanibel Island, Florida, USA, 2003, pp. 27-36.

[120] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, no. 2, pp. 51-53, June, 2007.

[121] Clark & Parsia. "*Pellet:The Open Source OWL2 Reasoner*," March 18, 2014 [online] Available: http://clarkparsia.com/pellet/.

[122] D. Tsarkov, and I. Horrocks, "FaCT++ description logic reasoner: System description," *Automated Reasoning*, U. Furbach and N. Shankar, eds., pp. 292-297, Heidelberg, Germany: Springer, 2006.

[123] B. Motik, R. Shearer, and I. Horrocks, "Hypertableau reasoning for description logics," *Journal of Artificial Intelligence Research,* vol. 36, no. 1, pp. 165-228, September, 2009.

[124] J. Kopena. "*OWLJessKB: a semantic web reasoning tool*," March 18, 2014 [online] Available: http://edge.cs.drexel.edu/assemblies/software/owljesskb/.

[125] C. J. Matheus, R. Dionne, D. F. Parent, K. Baclawski, and M. M. Kokar, "BaseVISor: A Forward-Chaining Inference Engine Optimized for RDF/OWL Triples," in *Proc. ISWC2006*, Athens, GA, 2006.

[126] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proc. WWW Alt. '04*, New York, NY, USA, 2004, pp. 74-83.

[127] M. Jang, and J.-C. Sohn, "Bossam: an extended rule engine for OWL inferencing," *Rules and Rule Markup Languages for the Semantic Web*, G. Antoniou and H. Boley, eds., pp. 128-138, Heidelberg, Germany: Springer, 2004.

[128] B. Grosof, and C. Neogy. "*SweetRules Project*," March 18, 2014 [online] Available: http://sweetrules.semwebcentral.org/.

[129] The Apache Software Foundation. "*Reasoners and rule engines: Jena inference support*," March 18, 2014 [online] Available: http://jena.apache.org/documentation/inference/.

[130] B. Motik, and R. Studer, "KAON2–A Scalable Reasoning Tool for the Semantic Web," in *Proc. ESWC2005*, Heraklion, Greece, 2005.

[131] Information Process Engineering (IPE), Institute of Applied Informatics and Formal Description Methods (AIFB), and Information Management Group (IMG). "*KAON2-Ontology Management for the Semantic Web*," March 18, 2014 [online] Available: http://kaon2.semanticweb.org/.

[132] Y. Zou, T. Finin, and H. Chen, "F-owl: An inference engine for semantic web," *Formal Approaches to Agent-Based Systems*, M. G. Hinchey, J. L. Rash, W. F. Truszkowski and C. A. Rouff, eds., pp. 238-248, Heidelberg, Germany: Springer, 2005.

[133] G. Meditskos, and N. Bassiliades, "Combining a DL reasoner and a rule engine for improving entailment-Based OWL reasoning," *The Semantic Web-ISWC 2008*, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin and K. Thirunarayan, eds., pp. 277-292, Heidelberg, Germany: Springer, 2010.

[134] G. Meditskos, and N. Bassiliades, "DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 8, no. 1, pp. 89-94, March, 2010.

[135] V. Haarslev, and R. Möller, "An empirical evaluation of optimization strategies for ABox reasoning in expressive description logics," in *Proc. DL'99,* Linköping, Sweden, 1999, pp. 115-119.

[136] I. Horrocks, L. Li, D. Turi, and S. Bechhofer, "The instance store: DL reasoning with large numbers of individuals," in *Proc. DL2004*, Whistler, British Columbia, Canada, 2004.

[137] B. Motik, and U. Sattler, "A comparison of reasoning techniques for querying large description logic aboxes," in *Proc. LPAR '06*, Phnom Penh, Cambodia, 2006, pp. 227-241.

[138] H. Li, Y. Wang, Y. Qu, and J. Z. Pan, "A Reasoning Algorithm for pD*," *The Semantic Web–ASWC 2006*, R. Mizoguchi, Z. Shi and F. Giunchiglia, eds., pp. 293-299, Heidelberg, Germany: Springer, 2006.

[139] S. Heymans, L. Ma, D. Anicic, Z. Ma, N. Steinmetz, Y. Pan, J. Mei, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, C. Feier, G. Hench, B. Wetzstein, and U. K. hide, "Ontology reasoning with large data repositories," *Ontology Management*, M. Hepp, P. D. Leenheer, A. D. Moor and Y. Sure, eds., pp. 89-128, New York, NY, USA: Springer, 2008.

[140] J. Broekstra, A. Kampman, and F. v. Harmelen, "Sesame: A generic architecture for storing and querying rdf and rdf schema," *The Semantic Web — ISWC 2002*, I. Horrocks and J. Hendler, eds., pp. 54-68, Heidelberg, Germany: Springer, 2002.

[141] Franz Inc. "*Introcuction of AllegroGraph 4.2*," March 18, 2014 [online] Available: http://www.franz.com/agraph/support/documentation/v4/agraph-introduction.html.

[142] The Apache Software Foundation. "*TDB*," March 18, 2014 [online] Available: http://jena.apache.org/documentation/tdb/.

[143] O. Erling, and I. Mikhailov, "Towards web scale RDF," in *Proc. SSWS2008*, Karlsruhe, Germany, 2008.

[144] O. Erling. "*Advances in Virtuoso RDF Triple Storage (Bitmap Indexing)*," March 18, 2014 [online] Available: http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSBitmapIndexing.

[145] B. Motik, "Reasoning in description logics using resolution and deductive databases," University of Karlsruhe, University of Karlsruhe, Karlsruhe, Germany, 2006.

[146] B. Motik, U. Sattler, and R. Studer, "Query answering for OWL-DL with rules," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 3, no. 1, pp. 41-60, July, 2005.

[147] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas, "The summary abox: Cutting ontologies down to size," *The Semantic Web - ISWC 2006*, I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold and L. M. Aroyo, eds., pp. 343-356, Heidelberg, Germany: Springer, 2006.

[148] J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, E. Schonberg, K. Srinivas, and L. Ma, "Scalable semantic retrieval through summarization and refinement," in *Proc. AAAI '07*, Vancouver, British Columbia, Canada, 2007, pp. 299-304.

[149] Y. Guo, and J. Heflin, "A scalable approach for partitioning owl knowledge bases," in *Proc. SSWS2006*, Athens, GA, USA, 2006, pp. 47–60.

[150] S. Wandelt, and R. Möller, "Scalability of OWL reasoning: role condensates," *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, R. Meersman, Z. Tari and P. Herrero, eds., pp. 1145-1154, Heidelberg, Germany: Springer, 2007.

[151] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "DL-Lite: Tractable description logics for ontologies," in *Proc. AAAI2005*, Pittsburgh, Pennsylvania, USA, 2005, pp. 602-607.

[152] D. Calvanese, D. Lembo, M. Lenzerini, and R. Rosati, "Data complexity of query answering in description logics," in *Proc. KR2006*, Lake District, UK, 2006, pp. 260–270.

[153] U. Hustadt, B. Motik, and U. Sattler, "Data complexity of reasoning in very expressive description logics," in *Proc. IJCAI'05*, Edinburgh, Scotland, UK, 2005, pp. 466-471.

[154] T. Eiter, G. Gottlob, M. Ortiz, and M. Simkus, "Query answering in the description logic Horn-SHIQ," *Logics in Artificial Intelligence*, S. Hölldobler, C. Lutz and H. Wansing, eds., pp. 166–179, Heidelberg, Germany: Springer, 2008.

[155] M. Schmidt, T. Hornung, M. Meier, C. Pinkel, and G. Lausen, "SP^{2}Bench: A SPARQL Performance Benchmark," *Semantic Web Information Management*, R.

d. Virgilio, F. Giunchiglia and L. Tanca, eds., pp. 371-393, Berlin, Heidelberg: Springer-Verlag, 2010.

[156] The Apache Software Foundation. "*ARQ - A SPARQL Processor for Jena*," March 18, 2014 [online] Available: https://jena.apache.org/documentation/query/.

[157] C. Bizer, and A. Schultz, "The berlin sparql benchmark," *International Journal on Semantic Web & Information Systems,* vol. 5, no. 2, pp. 1-24, 2009.

[158] semanticweb.org. "*SPARQL endpoint*," March 18, 2014 [online] Available: http://semanticweb.org/wiki/SPARQL_endpoint.

[159] W3C. "*SparqlEndpoints*," March 18, 2014 [online] Available: http://www.w3.org/wiki/SparqlEndpoints.

[160] S. Harris, N. Lamb, and N. Shadbolt, "4store: The Design and Implementation of a Clustered RDF store," in *Proc. SSWS2009*, Washington DC, USA, 2009, pp. 81-96.

[161] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 3, no. 2-3, pp. 158-182, October, 2005.

[162] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu, "Towards a complete OWL ontology benchmark," *The Semantic Web: Research and Applications*, Y. Sure and J. Domingue, eds., pp. 125-139, Heidelberg, Germany: Springer, 2006.

[163] T. Weithöner, T. Liebig, M. Luther, and S. Böhm, "What's wrong with OWL benchmarks," in *Proc. SSWS2006*, Athens, GA, USA, 2006, pp. 101–114.

[164] T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. v. Henke, and O. Noppens, "Real-world reasoning with OWL," *The Semantic Web: Research and Applications*, E. Franconi, M. Kifer and W. May, eds., pp. 296-310, Heidelberg, Germany: Springer, 2007.

[165] J. Urbani, F. v. Harmelen, S. Schlobach, and H. Bal, "QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases," *The Semantic Web–ISWC 2011*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy and E. Blomqvist, eds., pp. 730-745, Heidelberg, Germany: Springer, 2011.

[166] A. Yaseen, K. Maly, S. Zeil, and M. Zubair, "Performance Evaluation of Oracle Semantic Technologies With Respect To User Defined Rules," in *Proc. (in preparation for) International conference on Intelligent Semantic Web-Services and Applications*, Amman, Jordan, 2011.

[167] T. Weith¨oner, T. Liebig, M. Luther, and S. B¨ohm, "What's wrong with OWL benchmarks," in *Proc. Proceedings of the Second International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2006, pp. 101-114.

[168] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, "Application of hash to data base machine and its architecture," *New Generation Computing,* vol. 1, no. 1, pp. 63-74, March, 1983.

[169] Ontotext. "*owl-sameAs-optimization*," March 18, 2014 [online] Available: http://www.ontotext.com/owlim/owl-sameas-optimisation.

[170] H. Tamaki, and T. Sato, "OLD resolution with tabulation," in *Proc. ICLP1986*, London, United Kingdom, 1986, pp. 84-98.

[171] K. Marriott, and P. J. Stuckey, *Programming with constraints: an introduction*, Cambridge, Massachusetts: MIT press, 1998.

[172] J. Santos, and S. Muggleton, "When does it pay off to use sophisticated entailment engines in ILP?," *Inductive Logic Programming*, P. Frasconi and F. A. Lisi, eds., pp. 214-221, Heidelberg, Germany: Springer, 2011.

[173] R. Kowalski, and D. Kuehner, "Linear resolution with selection function," *Artificial Intelligence,* vol. 2, no. 3-4, pp. 227-260, Winter, 1972.

[174] B. Bishop, and F. Fischer, "Iris-integrated rule inference system," in *Proc. ARea2008*, Tenerife, Spain, 2008.

[175] J. Lobo, J. Minker, and A. Rajasekar, *Foundations of disjunctive logic programming*, Cambridge, Massachusetts: MIT press, 1992.

[176] F. Bry, N. Eisinger, T. Eiter, T. Furche, G. Gottlob, C. Ley, B. Linse, R. Pichler, and F. Wei, "Foundations of rule-based query answering," *Reasoning Web*, G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P.-L. Patranjan and R. Tolksdorf, eds., pp. 1-153, Heidelberg, Germany: Springer-Verlag, 2007.

[177] N. Leone, S. Perri, and F. Scarcello, "Improving ASP instantiators by join-ordering methods," *Logic Programming and Nonmotonic Reasoning*, T. Eiter, W. Faber and M. l. Truszczyński, eds., pp. 280-294, Heidelberg, Germany: Springer, 2001.

[178] J. Wielemaker, "An optimised semantic web query language implementation in prolog," *Logic Programming*, M. Gabbrielli and G. Gupta, eds., pp. 128-142, Heidelberg, Germany: Springer, 2005.

[179] I. Kollia, B. Glimm, and I. Horrocks. "*Answering queries over owl ontologies with sparql*," March 18, 2014 [online] Available: http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2011/KoGH11c.pdf.

[180] J. Struyf, and H. Blockeel, "Query optimization in inductive logic programming by reordering literals," *Inductive Logic Programming*, T. Horváth and A. Yamamoto, eds., pp. 329-346, Heidelberg, Germany: Springer, 2003.

[181] J. W. Lloyd, *Foundations of Logic Programming*, 2nd ed., Heidelberg, Germany: Springer, 1987.

[182] H. Shi, K. Maly, and S. Zeil, "Optimized Backward Chaining Reasoning System for a Semantic Web," in *Proc. WIMS '14*, Thessaloniki, Greece, 2014.

[183] The Apache Software Foundation. "*SDB - persistent triple stores using relational databases*," March 18, 2014 [online] Available: http://jena.apache.org/documentation/sdb.

[184] M. Klein, "Change management for distributed ontologies," Vrije Universiteit Amsterdam, 2004.

[185] S. N. Goodman, "Toward evidence-based medical statistics. 1: The P value fallacy," *Annals of internal medicine,* vol. 130, no. 12, pp. 995-1004, June, 1999.

VITA

Hui Shi

E&CS building, Department of Computer Science, Old Dominion University, Norfolk, VA 23529

Education

Sep.1999 - July 2003 B.S., major in Computer Science and Technology
School of Computer & Information
Hefei University of Technology, China

Sep.2003 - June 2006 M.S., major in Computer Application Technique
School of Computer & Information
Hefei University of Technology, China

Aug.2009 - Aug.2014 Ph.D., major in Computer Science
Department of Computer Science, Old Dominion University

Professional Experience: Teaching Experience:

2013.09- 2013.12: Instructor, Department of Computer Science, Old Dominion University, Norfolk, VA

2009.09- 2014.05: Teaching Assistant, Department of Computer Science, Old Dominion University, Norfolk, VA

2011.09- 2011.12: Lab Instructor, Department of Computer Science, Old Dominion University, Norfolk, VA

2006.06- 2009.08: Teaching Assistant, Hefei University of Technology, Hefei, Anhui, China